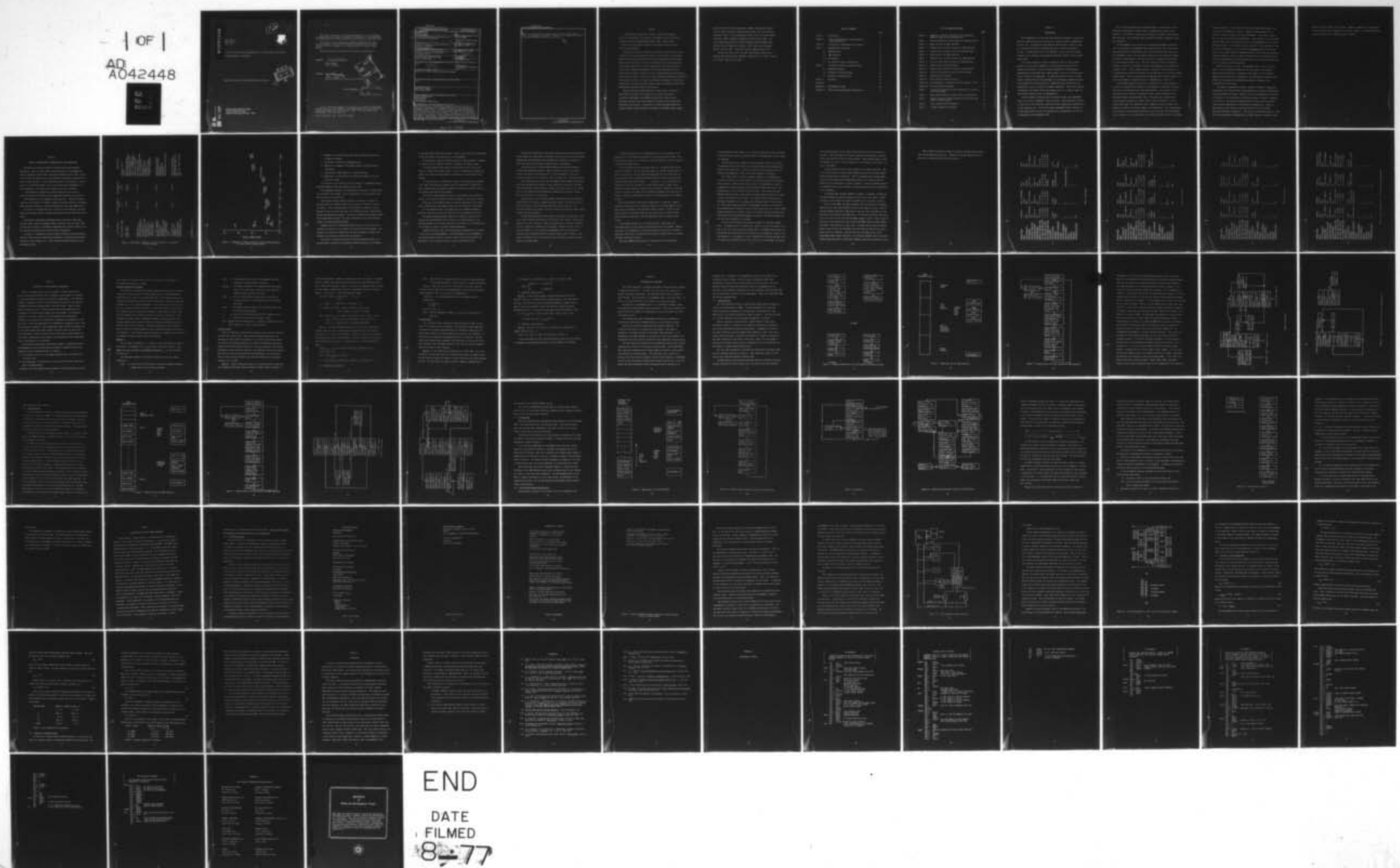
AD-A042 448 CLARKSON COLL OF TECHNOLOGY POTSDAM N Y DEPT OF ELEC--ETC F/G 17/2
A STUDY OF MICROPROCESSOR IMPLEMENTATION OF TIME REFERENCE DIST--ETC(U)
JUN 77 J F BOGDANOWICZ, K R KRISHNAN F30602-75-C-0082
UNCLASSIFIED RADC-TR-77-20 NL

1 OF 1
AD A042448



END
DATE
FILMED
8-77

RADC-TR-77-20
Phase Report
June 1977

A STUDY OF MICROPROCESSOR IMPLEMENTATION OF TIME REFERENCE DISTRIBUTION

Clarkson College of Technology

Approved for public release; distribution unlimited.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report contains a few illustrations which are not of the highest printing quality but because of economical consideration, it was determined in the best interest of the government that they be used in this publication.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED: Jacob Scherer

JACOB SCHERER
Project Engineer

APPROVED: Joseph J Naresky

JOSEPH J. NARESKY
Chief, Reliability & Compatibility Division

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DAP) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
18 1. REPORT NUMBER RADC-TR-77-20	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
6 4. TITLE (and Subtitle) A STUDY OF MICROPROCESSOR IMPLEMENTATION OF TIME REFERENCE DISTRIBUTION.		9 5. TYPE OF REPORT & PERIOD COVERED Phase Report
		6. PERFORMING ORG. REPORT NUMBER N/A
10 7. AUTHOR(s) Julius F. Bogdanowicz, David A. Perreault Komandur R. Krishnan Rangaswamy Mukundan		15 8. CONTRACT OR GRANT NUMBER(s) F30602-75-C-0082 63203F
9. PERFORMING ORGANIZATION NAME AND ADDRESS Clarkson College of Technology/Department of Electrical and Computer Engineering Potsdam NY 13676		16 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 95670015 17 QQ
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (RBC) Griffiss AFB NY 13441		11 12. REPORT DATE June 1977
		13. NUMBER OF PAGES 77
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same 12 85p.		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer:
Jacob Scherer (RBC)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Digital Communications
Microprocessors
Time Reference
Electromagnetic Compatibility

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The dissemination of a Time Reference throughout a network is useful in the operation of a switched digital communication network. The Time Reference Distribution Method (TRD) is a hierarchical scheme of dynamically allocating information paths for the flow of such reference information from the master node to all the other nodes in the network. This report is a detailed study of the feasibility of using a currently available microprocessor at each node to implement the TRD algorithm. The characteristics of some current microporcessors are studied. Three different implementation algorithms are developed for the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

over

406 944

LB

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

dynamic allocation process. A sample design of microprocessor system for one of the algorithms is presented. A estimate of the computing power needed is made and implementation problems are discussed.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

PREFACE

This effort was conducted by Clarkson College of Technology under the sponsorship of the Rome Air Development Center Post-Doctoral Program for Defense Communications Agency (DCA). Harris Stover of DCA was the task project engineer and provided overall technical direction and guidance.

The RADC Post-Doctoral Program is a cooperative venture between RADC and some sixty-five universities eligible to participate in the program. Syracuse University (Department of Electrical and Computer Engineering), Purdue University (School of Electrical Engineering), Georgia Institute of Technology (School of Electrical Engineering), and State University of New York at Buffalo (Department of Electrical Engineering) act as prime contractor schools with other schools participating via sub-contracts with the prime schools. The U.S. Air Force Academy (Department of Electrical Engineering), Air Force Institute of Technology (Department of Electrical Engineering), and the Naval Post Graduate School (Department of Electrical Engineering) also participate in the program.

The Post-Doctoral Program provides an opportunity for faculty at participating universities to spend up to one year full time on exploratory development and problem-solving efforts with the post-doctorals splitting their time between the customer location and their educational institutions. The program is totally customer-funded with current projects being undertaken for Rome Air Development Center (RADC), Space and Missile Systems Organization (SAMSO), Aeronautical Systems Division (ASD), Electronic Systems Division (ESD), Air Force Avionics Laboratory (AFAL), Foreign Technology Division (FTD), Air Force Weapons Laboratory (AFWL), Armament Development and Test Center (ADTC), Air Force Communications Service (AFCS), Aerospace Defense Command (ADC), Hq USAF, Defense Communications Agency (DCA), Navy, Army, Aerospace Medical Division (AMD), and Federal Aviation Administration (FAA).

Further information about the RADC Post-Doctoral Program can be obtained from Jacob Scherer, RADC/RBC, Griffiss AFB, NY, 13441, telephone AV 587-2543, COMM (315) 330-2543.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

Chapter 1

INTRODUCTION

The dissemination of an accurate Time Reference throughout a network has important applications in the operation of a switched digital communication network [1-4]. The method of Time Reference Distribution, studied in [5-8], is a hierarchical method of disseminating such a reference throughout a network. It enables each node of the network to select the best available reference in the network, by processing information received from its neighboring nodes.

This report presents a detailed feasibility study of using a microprocessor at each node to implement the operational procedure of Time Reference Distribution. Three alternative algorithms of microprocessor implementation are proposed and a sample design is given of a microprocessor system for one of the algorithms. It should be pointed out that no actual hardware simulations have been performed, but the design considerations have been presented in sufficient detail to demonstrate the feasibility of the design and to serve as a guide for hardware simulation. The study is thus an intermediate step between theory and hardware, and, it is hoped, brings the Time Reference method closer to implementation.

Various hardware techniques such as hard-wired logic and programmable logic can be used to implement a nodal processor. Hard-wired logic consists of logic gates, multivibrators, counters and other commercially available SSI or MSI components [9]. On the other hand, programmable logic consists of a stored program which controls the functions of a microprocessor (or CPU) in a minicomputer or microcomputer [10].

For a particular application, hard-wired logic, microprocessors, microcomputers or minicomputers could be used to implement the logical control functions. The tradeoffs between these techniques are based on developmental costs and time, ease of making system engineering changes, complexity, speed, and reliability.

The developmental time and cost for designing and debugging hard-wired logic in comparison to programmable logic restricts their use to high volume applications where the engineering costs can be distributed over many units. For programmable logic, once the logical functions of the design are defined, it becomes a simple matter of encoding them into a sequence of instructions and storing them in memory [11]. The use of design aids such as compilers, assemblers, editors, and simulators allows rapid debugging of the application programs [11,12]. System changes, whether the addition of new functions or the modification of existing functions, are accomplished via a relatively simple program change. With the hard-wired logic approach, minor logical changes can cause major hardware changes which must be implemented at a high cost.

A few remarks might be useful here to explain why the study considered the use of a microprocessor instead of a minicomputer to implement the logical functions of the Time Reference Distribution Method. The emergence of the microprocessor has enabled an impressive amount of data processing to be performed on small and inexpensive chips. This means that instead of the computing power residing in a single, large, central processor, it can be diffused into small and inexpensive units composed of groups of chips, each designed to match the particular needs of the application. This 'decentralization' results in an increased reliability, since the failure of an individual unit is likely to be less extensive in its effects than the failure of a single central processor. It also makes for easier and cheaper maintenance, since the chips are inexpensive to replace. However, microprocessors are, at present, a good deal slower [13] than minicomputers. One expects, however, that the gap will get narrower and narrower. Some estimates of the processing time are given later in the report (p.53) for the implementation of the Time Reference scheme. It is expected that the execution of the algorithm of Time Reference Distribution (i.e. the process of <u>reference</u> selection) will take place at much longer intervals (perhaps minutes or hours) than the exchange of timing information. Hence the speed of execution of the algorithm is not a critical consideration, and the estimated times on p. 53 are much smaller than will be required in network operation.

The microprocessor is still a technological infant. Hence, its full impact on data-processing problems and the full range of its potential applications are difficult to envisage at this stage of its development. However, the review articles appearing in the IEEE Spectrum (January, April 1976) describe applications already found and give a glimpse of the possibilities for the future.

The report is organized as follows. Chapter 2 contains a comparative description of the characteristics and capabilities of several microprocessors that are available at present. Chapter 3 is a brief explanation of the rules of operation of the Time Reference Distribution method. Chapter 4 examines the details of microprocessor implementation of those rules and presents three different algorithms of microprocessor implementation. Chapter 5 discusses the actual design of nodal processor for one of the three algorithms developed in Chapter 4. This is done in sufficient detail to show that the microprocessor implementation is indeed feasible. Chapters 4 and 5 comprise the main results of this study. Chapter 6 summarizes the conclusions of the study and offers suggestions for further studies. The Appendix contains a program listing for the sample design of Chapter 5.

Chapter 2

SURVEY OF MICROPROCESSOR CHARACTERISTICS AND CAPABILITIES

Microprocessors and microcomputers have been given many different definitions. Here, we shall define a microprocessor as a processing unit which is composed of one or more large-scale integration (LSI) chips, is able to accept data and modify it by using arithmetic and logical functions and is able to output the data. In this definition there is no reference to how the process is controlled, whether it is by macroinstruction programming or by microprogramming. This aspect lies outside of the basic definition.

A microcomputer is defined as a microprocessor which is used as a central processing unit (CPU) to which memory and I/O devices are connected.

Microprocessors can be compared in many ways [15]. They can be characterized by their speed, I/O transfer, number of internal registers, addressing modes, memory-accessibility, interrupt capabilities, and stack capabilities.

The speed of a microprocessor is a function of the technology, data and address path widths, number of separate paths and overlap in fetch and execute cycles.

Semiconductor processing technologies used include metal oxide semiconductor (MOS), silicon-on-sapphire (SOS), integrated injection logic (I^2L), and low-power Schottky. A performance comparison [16] is made in Table 1 for LSI processors manufactured by these different technologies.

In Figure 1 a comparison is made of memory efficiency versus operational speed for some presently available microprocessors, evaluating these devices through five test programs [17]. These benchmark programs consisted of the following routines:

LSI Technology	Gate Propagation Delay	Microinstruction Time	Applications
Bulk standard p-MOS	< 1 μs	100 μs	Calculator micro control
n-MOS (1974) C-MOS (1974)	< 100 ns	1-10 μs 2-10 μs	Medium speed and non-real-time data processing. Analog-to-digital conversion. Consumer controls. Low-speed monitor. High-speed calculator.
n-MOS (1975) C-MOS-on-sapphire I^2L (non-ion-implant) CDI (collector diffusion isolation) RTL (resistor-transistor logic) 3D-EFL (triple-diffused emitter-follower logic)	100-25 ns	0.5-1 μs	Stand-alone control Slow real-time processing. Fast a-d conversion. Fast monitor. Telephone terminal processing. Switching controls. Instrument management.
I^2L (ion-implanted) Low-power Schottky TTL (non-ion-implanted) Ion-implanted 3D-EFL C^3L (complementary constant-current logic)	25-10 ns	50-100 ns	High-speed controller. General-purpose data processing. Small mainframe controller. Delayed signal processing.
I^2L (1976) (ion-implanted plus Schottky) Low-power Schottky (ion-implanted) Emitter-coupled logic	10-1 ns	10-50 ns	Very fast controller. On-line process control. Big mainframe memory control. Real-time signal processing.

Table 1 - Performance comparison for LSI processors as a function of the technology

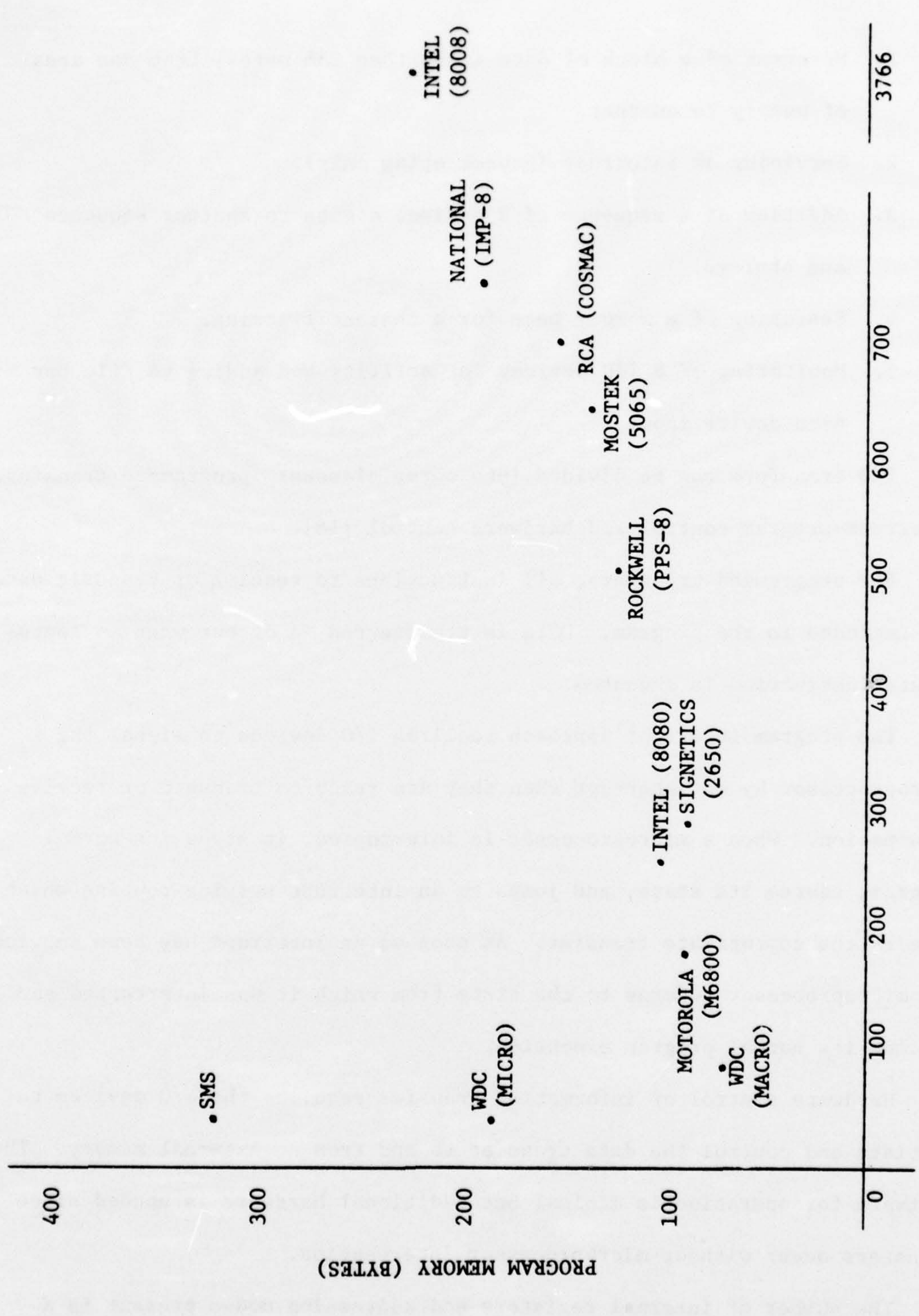


Figure 1 - Comparison of memory efficiency versus operational speed for some current microprocessors

1. Movement of a block of data (less than 256 bytes) from one area of memory to another.
2. Servicing an interrupt (housekeeping only).
3. Addition of a sequence of N decimal digits to another sequence and storage.
4. Searching of a memory page for a character string.
5. Monitoring of 8 I/O devices for activity and adding to file for each device input.

I/O transfers can be divided into three classes: programmed transfer, interrupt-program control and hardware control [18].

For programmed transfers, all instructions to receive or transmit data are included in the program. Data is transferred in or out when an appropriate instruction is executed.

The program interrupt approach requires I/O devices to signal the microprocessor by an interrupt when they are ready to transmit or receive information. When a microprocessor is interrupted, it stops its normal program, stores its state, and jumps to an interrupt service routine which effects the appropriate transfer. As soon as an interrupt has been serviced, the microprocessor returns to the state from which it was interrupted and resumes its normal program execution.

Hardware control of information transfer requires the I/O devices to initiate and control the data transfer to and from an external memory. The software for operation is minimal but additional hardware is needed since transfers occur without microprocessor intervention.

The number of internal registers and addressing modes present in a microprocessor is related to the speed of operation and also to the amount of external memory that will be needed. These factors also give an indication of how efficiently a microprocessor can be programmed.

Microprocessors have many different kinds of addressing modes. Examples are pointer, direct, indirect, relative, immediate, and indexed modes.

The pointer-address mode allows a microprocessor with a short work length to address large memory arrays. This is accomplished by keeping the address in a special register which is preloaded by an instruction in the program.

For any processor, an instruction consists of an opcode and an operand code. When using the direct-address mode, the operand code contains the direct address and the processor executes the instruction with data found in the location specified by that address.

The indirect-addressing mode results in the operand code containing a pointer to a location in memory in which the address of the data is located.

Using the relative-address mode, the address contained in the operand code is modified by a base address before the data location is referenced.

For the immediate mode, the operand code is an immediate address by which the processor executes the instruction on the operand code itself.

The indexed-addressing mode results in the address contained in the operand code being modified by the contents of an index register. This modified address not only allows any location in memory to be addressed, but it also makes it easier to perform repeated operations.

Another distinction that can be made between microprocessors is whether or not they are microprogrammable. Microprogramming is the programming of bit patterns which reside in a special control that directly controls the operation of each functional element in that microprocessor [19].

The interrupt capabilities vary greatly between different microprocessors. These capabilities range from no interrupts to multi-level vectored interrupts. Although most microprocessors have single-level interrupts, the method by which the state of the machine is saved can vary considerably.

Saving the state of an Intel 8008 microprocessor requires extensive software and additional hardware. When an interrupt occurs, the program counter is saved in an internal stack. If the accumulator or any other particular register is to be saved, it must be loaded into an external memory.

Some microprocessors such as the Intel 8080, Motorola M6800 and National PACE have alleviated this problem. Their architecture includes a push-down stack with specific instructions for saving the processor status. The stacks for the Intel 8080 and Motorola M6800 are implemented in external storage, while the PACE has an on chip stack.

Microprocessors such as the Intel 8080, Mostek 5065, Motorola M6800 and Rockwell PPS-8 can enable or disable interrupt requests with special instructions which set or reset an internal interrupt control flipflop. For processors without this ability, the use of external gating may provide similar control over the interrupts.

The PACE microprocessor can also enable or disable its interrupts. PACE uses a status register which uses 6 of its 16 bits for control. One bit can disable all of the interrupts. When an interrupt service routine is entered, this bit automatically disables all interrupts. However, this bit can be reset by software. Of the remaining programmable bits, one controls an interrupt that is generated when the internal stack is full or empty. This allows, through the use of an interrupt service routine, the stack to be extended in external memory.

If several interrupts occur simultaneously and it is necessary for one interrupt to be serviced before the other, a priority system is needed. Most microprocessors must rely on software or external hardware or both to establish a priority interrupt system.

Some microprocessors have priorities built in. The Mostek 5065 has two levels of interrupt. The four interrupt inputs of the PACE microprocessor are priority oriented, as are the eight interrupt inputs of the Toshiba TLCS-12. Microprocessors such as Fairchild's F8 and Rockwell's PPS-8 provide interrupt priorities since they have a daisy- structure. In this structure, all connections to the bus are made in a serial fashion where a signal can be modified by a device before it gets to the next device. Whenever an interrupt occurs in a device, the signal is not allowed to propagate past that point on the bus. Therefore, those devices closer to the microprocessor have a higher priority.

Most microprocessors only allow a single level of interrupt. When an interrupt occurs, there is a transfer of control to a certain memory location which gives the address of the interrupt service routine. If there is more than one source of possible interrupt, all sources must be polled to find which caused the interrupt so that the appropriate service routine can be called.

Intel's 8008 and 8080 use a vectored interrupt. When there is an interrupt request, several input bits called the vector are checked. Depending on the bit configuration, a different address for the service routine is specified. The vector is constructed with external hardware where up to eight interrupt sources are encoded.

Motorola's M6800 microprocessor in conjunction with its Peripheral

Interface Adaptor (PIA) allows, via a software routing, the control registers of each PIA to be polled on a priority basis to determine which device caused the interrupt.

Some microprocessors, however, do allow multiple-levels of interrupt. The Rockwell PPS-8, Toshiba TLCS-12, the Mostek 5065, and the National PACE exhibit these capabilities. The use of multiple level interrupt allows an immediate determination of which device has requested an interrupt.

The Toshiba TLCS-12 reserves a general register for a program-status word which contains the current state of the microprocessor and the program being executed. Whenever an interrupt occurs, the program-status word is exchanged with another program status word which defines the state of the microprocessor, so the interrupt can be serviced. Since there are eight separate interrupt lines, eight separate external memory locations are needed to store the exchanged status words. The highest priority interrupt uses location 8 in memory for its exchange and the lowest priority uses location 15. Since each interrupt line has an independent priority, an interrupt is only accepted when the mask bit located in the status word is a 1 and no higher priority interrupt is requested. After the interrupt has been serviced, the status word of the interrupted program is restored to the general register so the interrupted program can be resumed.

Microprocessors usually have either an internal or external pushdown stack. A pushdown stack is a linear list, that is, the last item added to the list is the first item that can be removed [20]. A pushdown stack allows the storing and later retrieving of the contents of the accumulator, flags, or any data register. One advantage of using a pushdown stack is that multiple level interrupts can be handled since on the occurrence of an interrupt, the status of the microprocessor can be saved and then restored after the interrupt is serviced. Other advantages are register transfers are minimized, sorting is aided, and subroutine nesting is made possible. When external memory is used to implement the stack, the only limitation on the length of the stack is the size of the memory.

Some microprocessors have a direct-memory access (DMA) capability. DMA is the rapid transfer of data between peripheral devices and external memory without microprocessor supervision. This is accomplished by stealing memory cycles from the program and transferring the data to or from locations in memory which are addressed by a special register. Since this address register is incremented after every transfer, successive data words can be transferred to or from memory.

To implement DMA, external hardware is needed. A register is needed for the memory address and another register is used for the word count. Control circuitry is also needed to initiate the memory cycle, once data is ready.

With the Intel 8080, the address register must be loaded with the memory address and the word count register with the total number of words. When the external hardware generates a HOLD input to the CPU, and if the CPU is in the HALT state, the CPU enters the HOLD state. The CPU will also enter this state if it is in the T2 or TW state and the READY signal is active. The T2 state is that part of the instruction cycle in which the READY, HOLD, and HALT signals are tested. The TW state is a wait state. The HOLD state allows an external device to gain control of the address and data busses as soon as the current machine cycle is completed. When the CPU enters the HOLD state, a hold acknowledge (HLDA) signal indicates to the external hardware that the data and address busses have gone to their high impedance state and a transfer can occur.

Table 2 gives an overview of some of the major characteristics found in some current microprocessors [15]. Appendix B lists the manufacturers of some of the currently available microprocessors.

MANUFACTURER	FAIRCHILD	GEN. INSTRUMENTS	INTEL	INTEL
MODEL	8	CP1600	3000	4004
CHIP TECHNOLOGY	NMOS	NMOS	BIPOLAR	PMOS
ARCHITECTURE	8 BIT PARALLEL	16 BIT PARALLEL	2 BIT SLICE	4 BIT PARALLEL
ALU/LOGIC SHARE CHIP	YES	YES	NO	YES
CLOCK FREQ./PHASES	2 MHZ/2 PH	5 MHZ/2 PH	8 MHZ/1 PH	750 KHZ/2 PH
MICROPROGRAMMED	YES	YES	USER	NO
NUMBER OF INSTRUCTIONS	101 (8 BIT)	68 (16 BIT)	VARIABLE	46 (8 BIT)
REG LOAD TIME FOR INSTR	2 μs (8 BIT)	2.4 μs (16 BIT)	150 ns (16 BIT)	10.8 μs (8 BIT)
ADD TIME (REG TO REG)	2 μs (8 BIT)	2.4 μs (16 BIT)	300 ns (16 BIT)	10.8 μs (8 BIT)
REG TO MEMORY ADD TIME	5 μs (8 BIT)	3.2 μs (16 BIT)	VARIABLE	10.8 μs (4 BIT)
INPUT/OUTPUT				
DATA PATH WIDTH	8 BITS	16 BITS	VARIABLE	4 BITS
INTERRUPTS	CHAIN	NESTED	VECTORED PRIORITY	NO
PERIPHERAL INTERFACES	--	--	--	DISPLAY
SOFTWARE				
RESIDENT ASSEMBLER	X	X	--	X
CROSS ASSEMBLER	X	X	CROSS MICROASSEMBLER	X
MONITOR	X	X	--	X
LANGUAGES	--	--	--	--
SIMULATOR	X	X	--	X
PROTOTYPING SYSTEMS	YES	--	--	YES

Table 2 (Continued)

MANUFACTURER	INTEL	INTEL	INTEL	MONOLITHIC
MODEL	4040	8008/8008-1	8080	6701
CHIP TECHNOLOGY	PMOS	PMOS	NMOS	BIPOLAR
ARCHITECTURE	4 BIT PARALLEL	8 BIT PARALLEL	8 BIT PARALLEL	4 BIT SLICE
ALU/LOGIC SHARE CHIP	YES	YES	YES	NO
CLOCK FREQ./PHASES	1 MHZ/2 PH	500/800 KHZ/2 PH	2 MHZ/2 PH	5 MHZ/1 PH
MICROPROGRAMMED	NO	NO	NO	USER
NUMBER OF INSTRUCTIONS	60 (8 BIT)	48 (8 BIT)	78 (8 BIT)	VARIABLE
REG LOAD TIME FOR INSTR	8 μs (8 BIT)	20/12.5 μs (8 BIT)	2.5 μs (8 BIT)	1.2 μs (16 BIT)
ADD TIME (REG TO REG)	8 μs (8 BIT)	20/12.5 μs (8 BIT)	2 μs (8 BIT)	900 ns (16 BIT)
REG TO MEMORY ADD TIME	8 μs (8 BIT)	32/20 μs (8 BIT)	3.5 μs (8 BIT)	1.2 μs (16 BIT)
INPUT/OUTPUT				
DATA PATH WIDTH	4 BITS	8 BITS	8 BITS	VARIABLE
INTERRUPTS	VECTORED	VECTORED	VECTORED	1 LEVEL, PRIORITY OPT
PERIPHERAL INTERFACES	DISPLAY	I/O PORT, GP	I/O PORT, GP	--
SOFTWARE				
RESIDENT ASSEMBLER	X	X	X	X
CROSS ASSEMBLER	X	X	X	X
MONITOR	X	X	X	X
LANGUAGES	--	PL/M	PL/M	--
SIMULATOR	X	X	X	--
PROTOTYPING SYSTEM	YES	YES	YES	YES

Table 2 (Continued)

MANUFACTURER	MOSTEK	MOTOROLA	NATIONAL SEMI	RCA
MODEL	5065	M6800	IMP-8A/500D	COSMAC
CHIP TECHNOLOGY	PMOS	NMOS	PHOS	CHOS
ARCHITECTURE	8 BIT PARALLEL	8 BIT PARALLEL	4 BIT SLICE	8 BIT PARALLEL
ALU/LOGIC SHARE CHIP	YES	YES	NO	YES
CLOCK FREQ./PHASES	1.4 MHZ/3 PH	1 MHZ/2 PH	715 KHZ/4 PH	2.67 MHZ/1 PH
MICROPROGRAMMED	NO	YES	USER	YES
NUMBER OF INSTRUCTIONS	51 (8/16 BIT)	72 (8 BIT)	38 (8 BIT)	59 (8 BIT)
REG LOAD TIME FOR INSTR	8.5 μs (8 BITS)	2 μs (8 BIT)	11.2 μs (8 BIT)	6 μs (8 BIT)
ADD TIME (REG TO REG)	10 μs (8 BIT)	2 μs (8 BIT)	4.2 μs (8 BIT)	18 μs (8 BIT)
REG TO MEMORY ADD TIME	10 μs (8 BIT)	2 μs (8 BIT)	11.2 μs (8 BIT)	6 μs (8 BIT)
INPUT/OUTPUT				
DATA PATH WIDTH	8 BIT	16 BIT	8 BIT	8 BIT
INTERRUPTS	PRIORITY	YES	YES	MASKABLE
PERIPHERAL INTERFACES	--	THROUGH ADAPTOR	TTY DISPLAY	--
SOFTWARE				
RESIDENT ASSEMBLER	X	--	X	--
CROSS ASSEMBLER	X	X	X	X
MONITOR	--	X	X	X
LANGUAGES	--	--	--	--
SIMULATOR	--	X	--	X
PROTOTYPING SYSTEM	NO	YES	YES	YES

Table 2 (Continued)

MANUFACTURER	ROCKWELL	ROCKWELL	SIGNETICS	WESTERN DIGITAL
MODEL	PPS-4	PPS-8	2650I "PIP"	CP 1611/1621/1631
CHIP TECHNOLOGY	PMOS	PMOS	NMOS	NMOS
ARCHITECTURE	4 BIT PARALLEL	8 BIT PARALLEL	8 BIT PARALLEL	8/16 BIT PARALLEL
ALU/LOGIC SHARE CHIP	YES	YES	YES	NO
CLOCK FREQ./PHASES	200 KHZ/4 PH	250 KHZ/4 PH	1.25 MHZ/1 PH	3.3 MHZ/4 PH
MICROPROGRAMMED	YES	YES	NO	USER
NUMBER OF INSTRUCTIONS	50 (8 BIT)	109 (8,16,24 BIT)	72 (8,16,24 BIT)	OVER 80 (16 BIT)
REG LOAD TIME FOR INSTR	4 s (8 BIT)	5 s (8 BIT)	4.8 s (8 BIT)	900 ns (8 BIT)
ADD TIME (REG TO REG)	4 s (8 BIT)	4 s (8 BIT)	4.8 s (8 BIT)	300 ns (8 BIT)
REG TO MEMORY ADD TIME	4 s (8 BIT)	5 s (8 BIT)	4.8 s (8 BIT)	1.2 s (8 BIT)
INPUT/OUTPUT				
DATA PATH WIDTH	4 BIT	8 BIT	8 BIT	8/16 BIT
INTERRUPTS	NO	3x16 DAISY CHAIN	1 LEVEL VECTORED	PRIORITY, 4 LEVEL
PERIPHERAL INTERFACES	DISPLAY,TTY,CP	DISPLAY,TTY,GP	--	--
SOFTWARE				
RESIDENT ASSEMBLER	X	X	--	--
CROSS ASSEMBLER	X	X	X	--
MONITOR	--	--	X	--
LANGUAGES	--	--	--	--
SIMULATOR	X	X	--	--
PROTOTYPING SYSTEM	YES	YES	NO	--

Table 2 - Major characteristics of some current microprocessors.

Chapter 3

THE METHOD OF TIME-REFERENCE DISTRIBUTION

There is a growing need [1-3] to disseminate a common reference for precise time and time-interval (PTTI) throughout large networks, for example, in such applications as navigation and digital communication. The "Transfer Standard" technique [4] uses a fixed hierarchical structure in the network to disseminate PTTI from the master node to the other nodes. However, fixed structure techniques are of limited use when the network operation has to accommodate clock failures and communication link outages. An alternative approach is to devise a scheme of self-organization which dynamically allocates the network path over which PTTI information is transmitted to each node [5,6]. This self-organization is accomplished by assigning a rank to each nodal clock and a demerit to each communication link, and then providing a set of rules which enable each node to decide over which link it should accept PTTI information. Earlier versions [5,6] of these rules, though adequate for the initial organization of the network, do not guarantee its re-organization after a perturbation of its structure.

The method of Time Reference Distribution (TRD) is a modified version of self-organization which has been shown to organize a network as well as re-organize it after a disturbance [8]. It ensures

(a) the automatic selection of the highest ranking node in the network as the master node.

(b) the transmission of information to each node from this master node by a path of minimum demerit.

In addition, when the network has been organized, each node knows its position with respect to the master node in the 'tree' along which information is disseminated throughout the network.

NETWORK ORGANIZATION TECHNIQUE

The nodes of the network are assigned unique ranks which reflect relative accuracies of their clocks. The links between nodes are assigned demerits which reflect the quality of the interconnection. The proposed technique is iterative, and assumes that between the k'th and (k+1)st iterations, each node transmits to all its connected neighbors: (a) its rank, (b) the rank of the node it is using as ultimate reference, (c) the total demerit of the path to its ultimate reference, and (d) the reading of a nodal update counter. Each node then applies a set of Selection Rules to items (a), (b) and (c) of the information received from its neighbors and makes a tentative choice of an ultimate reference and a link over which to receive the reference (unless it chooses itself as the reference). The node then applies a set of Decision Rules to item (d) to decide whether to use the selected reference or resort to self-reference. The Selection and Decision Rules are, together, sufficient for the network to organize itself under all conditions.

Notation

Let the nodes be numbered 1,...,n where n is the total number of nodes in the network. Furthermore, let r_i be the rank of the clock at node i. The higher the rank, the lower is the numerical value of r_i. No two nodes have the same rank.

The following variables are defined for each node i and for a given iteration, k.

$U_i(k)$ $\triangleq$ rank of the clock which node i uses as an ultimate reference between the k'th and (k+1)st iterations.

$I_i(k)$ $\triangleq$ the node which node i used as its immediate reference between the k'th and (k+1)st iterations.

$d_{ij}=d_{ji}$ $\triangleq$ the demerit assigned to the communications link between nodes i and j when such a link exists; the larger the numerical value, the worse the link.

$D_i(k)$ $\triangleq$ the total path demerit by which node i received the ultimate reference it uses between the k'th and (k+1)st iterations.

$T_i(k)$ $\triangleq$ the update counter at node i for the period between the k'th and (k+1)st iterations.

C_i $\triangleq$ the set of all nodes which are directly linked to node i. This set does not contain the node i itself.

Note: Symbols which are modified by a ^ indicate a tentative value for that variable, i.e., $\hat{U}_i(k)$, $\hat{I}_i(k)$ and $\hat{D}_i(k)$.

Selection Rules

There are three basic rules which are used to select the best tentative reference for a node to use. These rules are applied sequentially and determine the best tentative reference to use in the time interval between the (k+1)st and (k+2)nd iterations on the basis of information transmitted between the k'th and (k+1)st iterations. If a given selection rule uniquely determines the best tentative reference, the remaining rules are not applied. Once the best tentative reference has been determined, a set of Decision Rules are used to decide whether this tentative reference or an alternate reference is to be used.

Rule S1. A node i tentatively selects its reference from the link with that neighboring node which used the highest ranking ultimate reference in the previous iteration. However, if the rank of node i is equal to or greater than the ultimate rank used by the directly connected node (immediate node), node i tentatively references itself. If two or more immediate nodes reference the same highest ultimate rank, rule S1 is inconclusive and rule S2 must be applied. Stating S1 in a concise mathematical manner:

Let $\hat{U}_i(k) = \min\{r_i, \min_{j \in C_i} U_j(k-1)\}$ $\quad i=1,\ldots,n$

a) if $\hat{U}_i(k) = r_i$, then $\hat{I}_i(k) = i$ and $\hat{D}_i(k) = 0$;

b) otherwise, if $\hat{U}_i(k) = U_q(k-1)$, then let $\hat{I}_i(k) = q$ and
$\hat{D}_i(k) = D_q(k-1) + d_{ij}$ if q is unique.

Note: Rule S1 fails to uniquely select $\hat{I}_i(k)$ if two or more of the $U_j(k-1)$ terms are equal to the minimum value of $\hat{U}_i(k)$ and $\hat{U}_i(k) \neq r_i$. However, $\hat{U}_i(k)$ is uniquely determined.

Rule S2. From those immediate nodes which reference the same highest ultimate rank, the link is tentatively chosen which passes the reference information over a path of least demerit. However, if two or more dissemination paths have the same minimum demerit, rule S2 is inconclusive and rule S3 must be applied to those paths. Stating rule S2 in a concise mathematical manner:

Suppose $j_1,\ldots,j_v$ are the nodes which give a minimum value for $\hat{U}_i(k)$ in Rule S1.

Let: $\hat{D}_i(k) = \min_{1 \leq p \leq v}\{d_{ij} + D_{j_p}(k-1)\}$

a) If the minimum is achieved by a unique j_σ, then $\hat{I}_i(k) = j_\sigma$

b) Otherwise, apply Rule S3.

Note: Rule S2 fails to uniquely select $\hat{I}(k)$ if two or more paths have the same minimum demerit. However, $\hat{D}_i(k)$ is uniquely determined.

Rule S3. When rules S1 and S2 are inconclusive, the best tentative reference path is selected as the minimum demerit path that has the highest ranking immediate node. Stating rule S3 mathematically:

Suppose $j_1, \ldots, j_t$ are nodes which attain the minimum of Rule S2.

Suppose that

$$r_{j_q} = \min_{1 \leq p \leq t} \{r_{j_p}\}$$

Then let $I_i(k) = j_q$.

Note: Rule S3 determines a unique j_q, since no two clocks have the same rank.

Decision Rules

Once a tentative best reference has been selected by a node, it must decide if it should use that reference. This decision is made using three rules which are applied sequentially based on information supplied by the nodal update counter of the given node and the tentative best reference. If a given rule is satisfied, the remaining rules are not applied. Once the nodal clock system becomes organized, each node knows its position from the best reference since the update counter at each node specifies the number of links in the path to this reference.

Rule D1. If the tentative best reference for a given node i is a self-reference, the node uses itself as a reference and reduces its update counter by one unless its counter is already at zero in which case the counter remains at zero. If rule D1 does not apply, rule D2 is applied. Mathematically:

a) if $\hat{U}_i(k) = r_i$, then $U_i(k) = r_i$, $I_i(k) = i$, $D_i(k) = 0$, and

$$T_i(k) = \begin{cases} T_i(k-1) - 1 & \text{if } T_i(k-1)>0 \\ 0 & \text{if } T_i(k-1)=0 \end{cases}$$

b) otherwise, apply Rule D2.

Rule D2. If the received update counter associated with the best tentative reference is smaller than the update counter at the given node i, the node i uses the best reference received and makes its update counter equal to the received update counter from the best tentative reference incremented by one. If rule D2 does not apply, rule D3 is applied. Hence,

a) if $T_{\hat{I}_i(k)}(k-1) < T_i(k-1)$, then $U_i(k) = \hat{U}_i(k)$, $I_i(k) = \hat{I}_i(k)$,

$D_i = \hat{D}_i(k)$ and $T_i(k) = T_{\hat{I}_i(k)}(k-1) + 1$,

b) otherwise, apply Rule D3.

Rule D3. The node i uses itself as a reference and increments its update counter by one; i.e.,

$U_i(k) = r_i$, $I_i(k) = i$, $D_i(k) = 0$ and $T_i(k) = T_i(k-1) + 1$.

It has been shown [8], that these Decision Rules will cause the network to converge to an organized structure in a finite number of iterations.

Chapter 4

IMPLEMENTATION ALGORITHMS

The logical approach to designing programmed or hard-wired logic systems is similar in nature [21]. In either case, the first step consists of a general functional partitioning. For hard-wired logic, this is known as a block diagram. The counterpart for programmable logic is the flow chart. In either case, system definition can be made to any degree of detail.

One approach to programmable logic is to subdivide the general system into modules with well-defined inputs and outputs. This gives freedom in the manner by which the modules are implemented as long as the input and output constraints are met.

This approach was taken in programming the process of dynamically allocating a hierarchical information path in an interconnected system.

The process is based on comparisons made between iterations. The iteration time interval is divided into three phases. Each phase is initiated in the nodal processor by externally timed interrupts. It is assumed that the nodal processor is initially in a wait loop, waiting to be interrupted and it will return to this state upon completion of the interrupt.

The first phase (TR) constitutes the transmitting and receiving of nodal information by a node with its directly connected nodes. The second phase, (TE), ends the transmission and reception phase and provides a guard band of time between the two major phases. The third phase (TD), consists of a selection and decision process whereby the best nodal reference is determined for use in the time interval between this iteration and the next. In the TR phase, the nodal information from the connected nodes is inputted on an interrupt basis. In Figure 2, the general flow chart for each phase of the iterative process is shown. As can be seen, the method by which nodal information is stored from the channel buffers is not specified. Also, the method by which the selection and decision process is implemented is not specified. Three different algorithms have been developed, specifying different ways in which these processes could be implemented. These are called TMAB, TMNB, SM, and are explained below.

4.1. TMAB Algorithm

The first algorithm utilizes a storage table wherein nodal information is stored on an attribute basis (TMAB). Storing information in this manner entails storing all information of the same type in the same block. The memory map of the storage allocation is shown in Figure 3. The first location of each block contains the self-reference information of the node.

When an interrupt is received from a channel buffer to input nodal information, Figure 4, a flag is set in memory to indicate that information was received from that particular channel buffer. Information is stored in the proper blocks by using the channel buffer number as a link. Each time a channel buffer interrupt is received, all buffers are checked to determine if any other channels are also ready to input data. Hence, the time needed to initiate the interrupt service routine is minimized. For all algorithms, the path demerit is updated before being stored.

Once all nodal information is received and loaded into the storage table, the best tentative reference is selected. Then a decision is made to determine if this tentative reference should be used.

The TMAB selection and decision algorithm operates by checking only those storage locations for which a flag is set. The flag for the self-reference

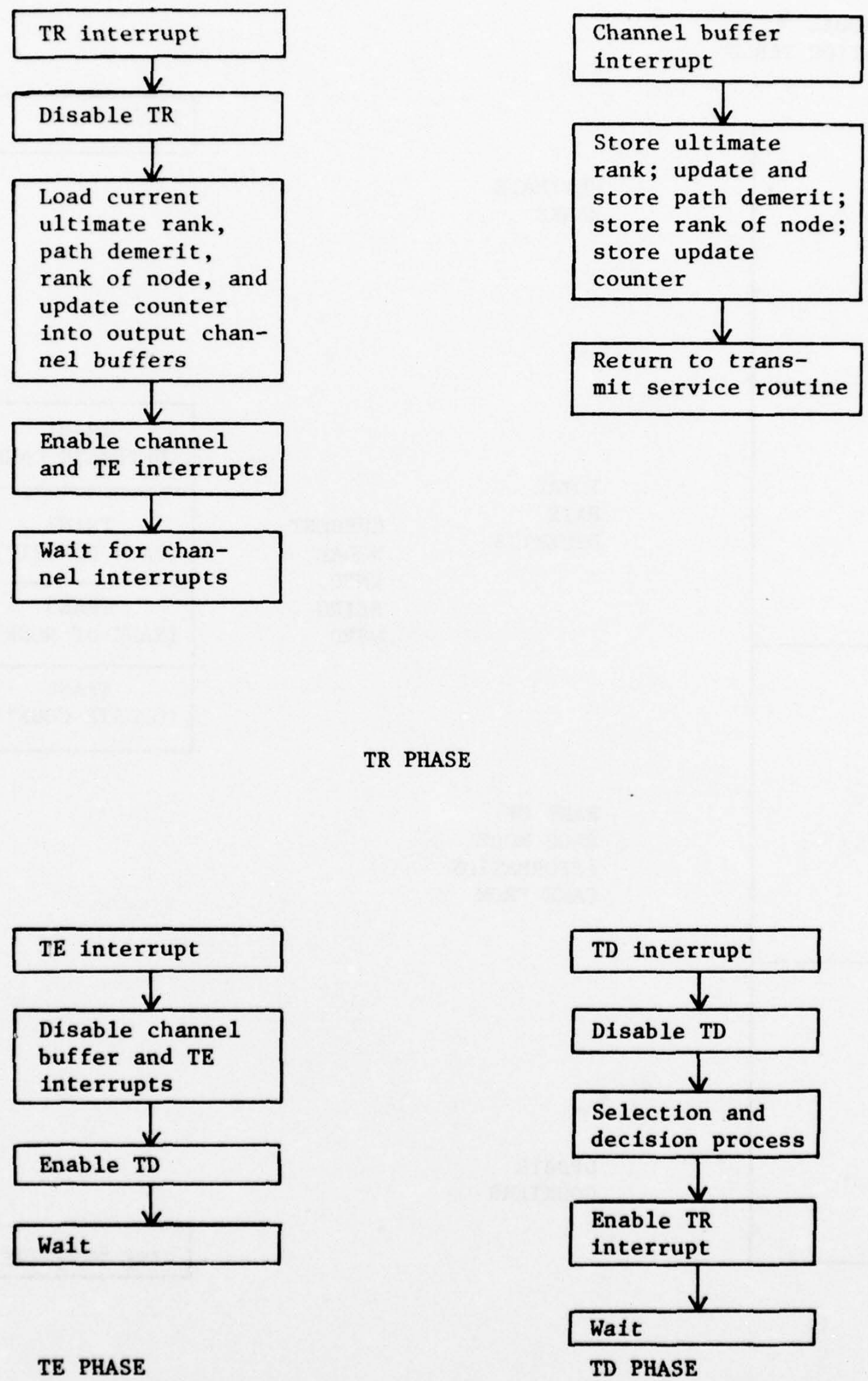


Figure 2 - General flowchart for the three phases of iteration process

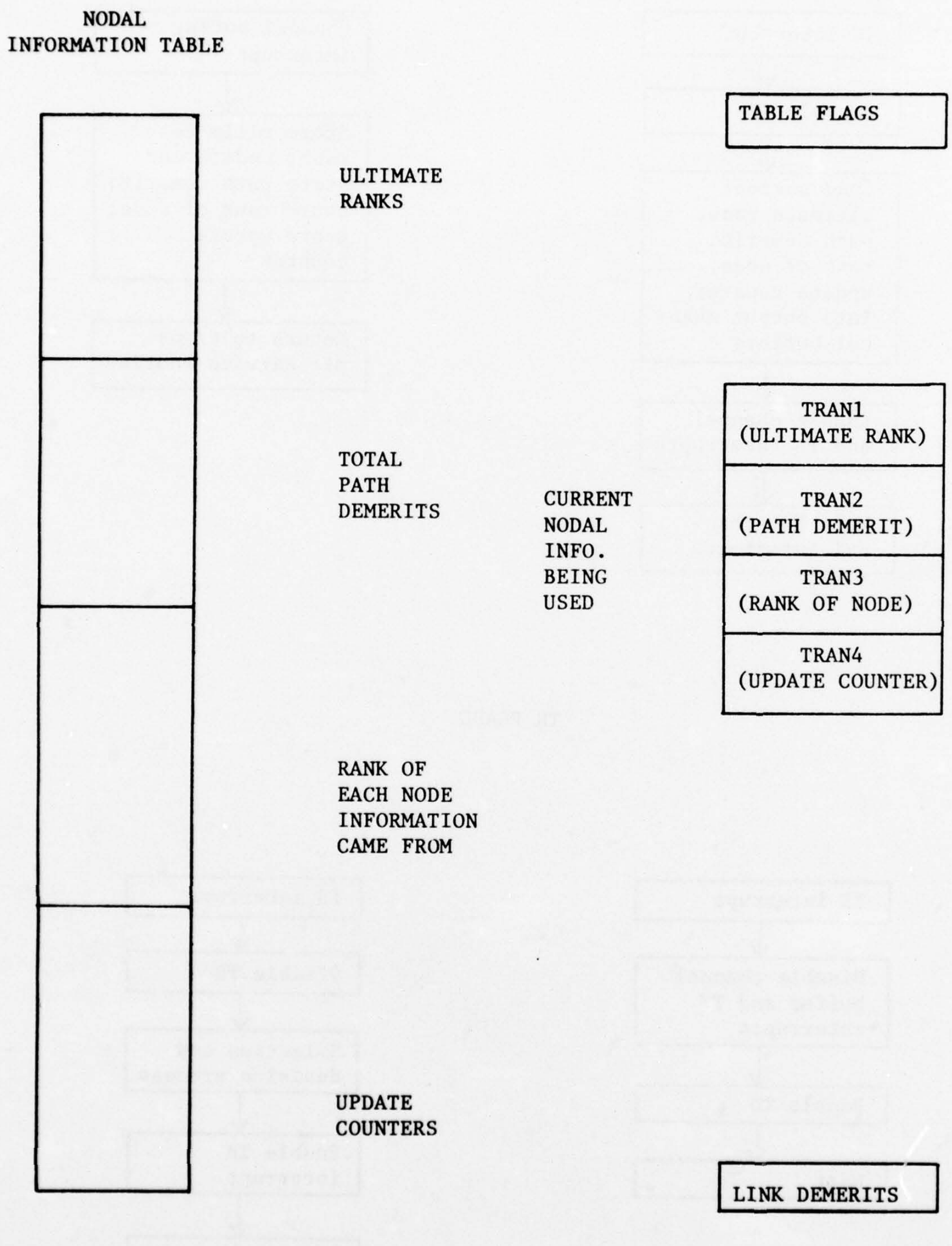


Figure 3 - Memory map for the TMAB algorithm

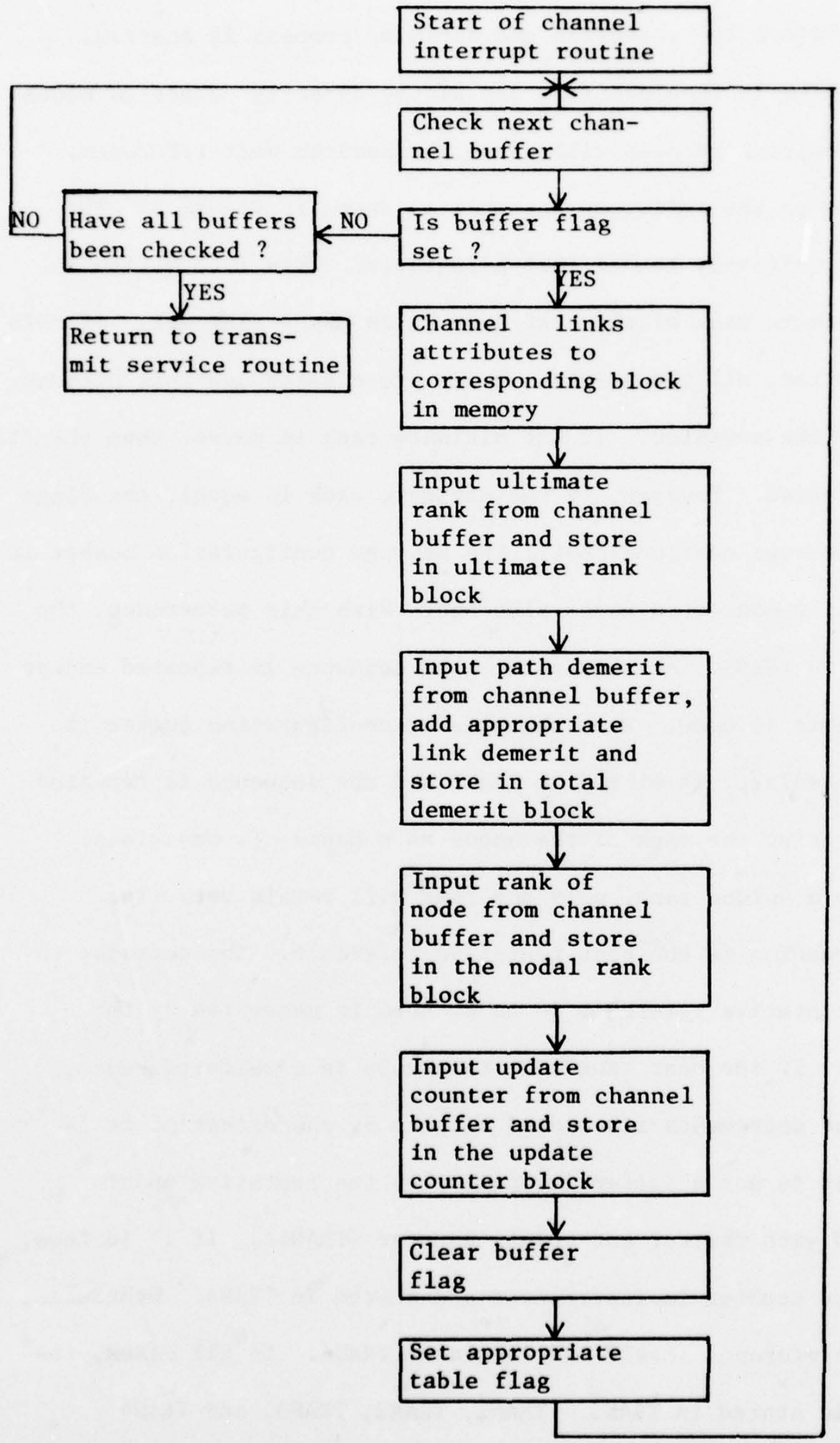


Figure 4 - Channel buffer interrupt routine for TMAB algorithm

information is set before the selection and decision process is started. Thus, if no information is received from any of the directly connected nodes, the selection and decision process will cause the node to self-reference. This algorithm works on the following sequence as shown in Figure 5. The rank of the node is initially loaded into a register. This information is compared to the ultimate rank of the next node which has a flag set. If this ultimate rank is better, all the previous flags are cleared and this ultimate rank is loaded into the register. If the ultimate rank is worse, then the flag for that node is cleared. However, if the ultimate rank is equal, the flags remain set. This process continues until the storage configuration number is the number of directly connected nodes plus one. With this occurrence, the register is stored in TRAN1. At this point, the sequence is repeated except the total path demerit is used. When the storage configuration number is reached again, the register is stored in TRAN2 and the sequence is repeated for the final time using the rank of the nodes as a basis of comparison. Since each node has a unique rank, only one flag will remain set. The corresponding information is the best tentative reference. The decision to determine if this tentative reference is to be used is generated by the following sequence. If the best tentative reference is a self-reference, the node uses it and decrements its update counter by one except if it is already zero. If it is not a self-reference, then the tentative update counter is compared with the current update counter (TRAN4). If it is less, the tentative update counter is incremented and stored in TRAN4. Otherwise, the node will self-reference itself and increment TRAN4. In all cases, the rank of this node is stored in TRAN3. TRAN1, TRAN2, TRAN3, and TRAN4 constitute the current information that will be transmitted to all connected

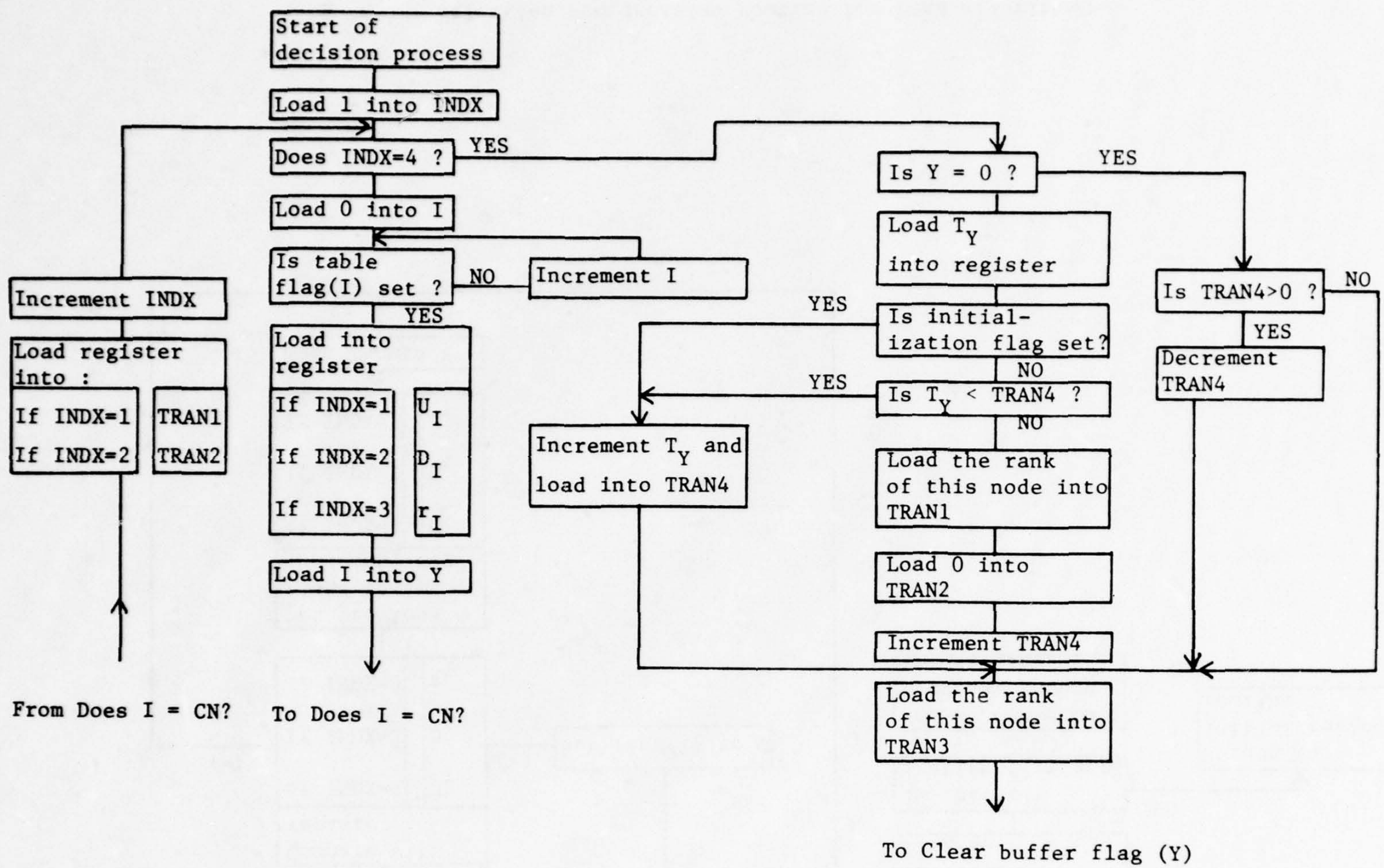


Figure 5 (Continued)

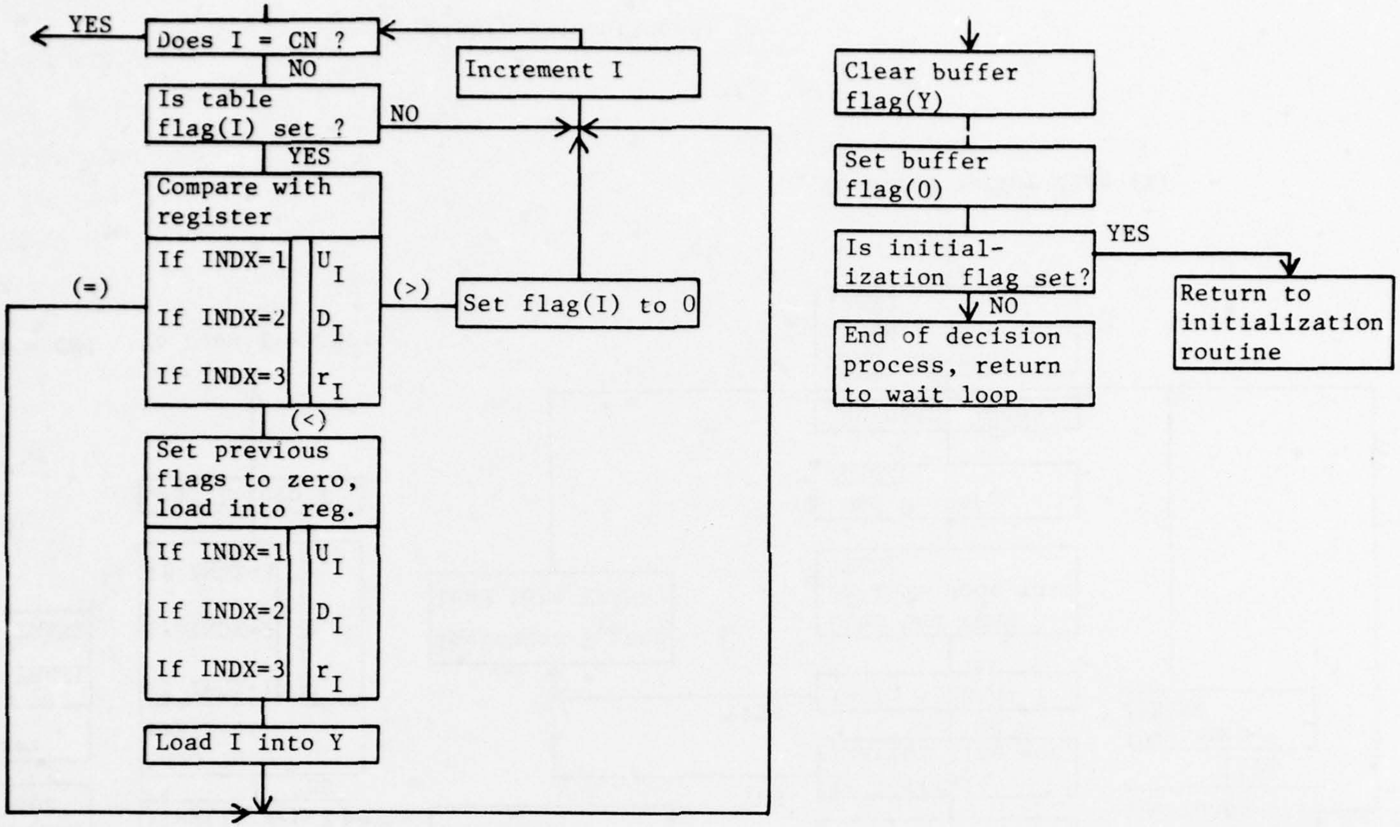


Figure 5 - Selection and decision routine for TMAB algorithm

nodes during the next iteration.

4.2. TMNB Algorithm

The second algorithm utilizes a storage table in which nodal information is stored on a node basis (TMNB). In the storage table, each connected node is allocated a storage block for storing its nodal information. The self-reference of this node is stored in block 0. During initialization, this information is stored in the tentative reference storage. The memory map of the storage allocation is shown in Figure 6.

The TMNB channel buffer interrupt routine, Figure 7, is similar to that used by the TMAB algorithm. The major difference is the form in which the nodal information is stored.

The TMNB selection and decision process, Figure 8, operates on the principle of comparing the information stored in the table with that stored in the tentative reference. This comparison is accomplished by comparing bytes until a unique decision is made determining whether the information stored in the tentative reference is better than the information received from the connected node. If the received information is better, it is stored in the tentative reference. A comparison is made with only those node blocks for which a flag is set. This comparison sequence is repeated until the storage configuration number is reached. Then, a decision is made to determine if the best tentative reference should be used. The implemented decision sequence is very similar to that used by the TMAB algorithm. The major difference is that the tentative and current nodal information is kept in separate blocks during the selection and decision process. The TMAB algorithm uses one block to store the tentative ultimate rank, tentative path demerit, rank of this node and current update counter. A flag indicates

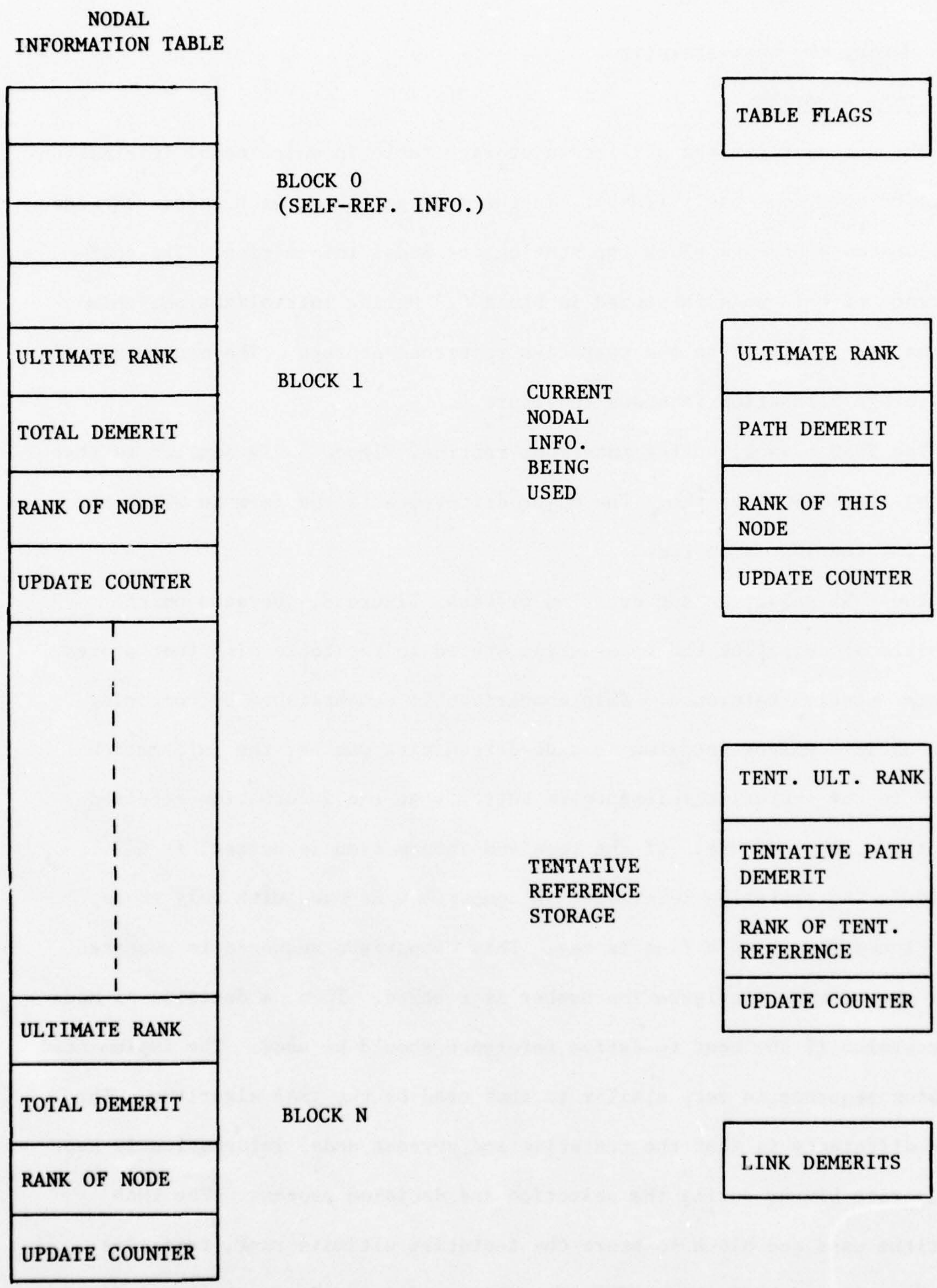


Figure 6 - Memory map for the TMNB algorithm

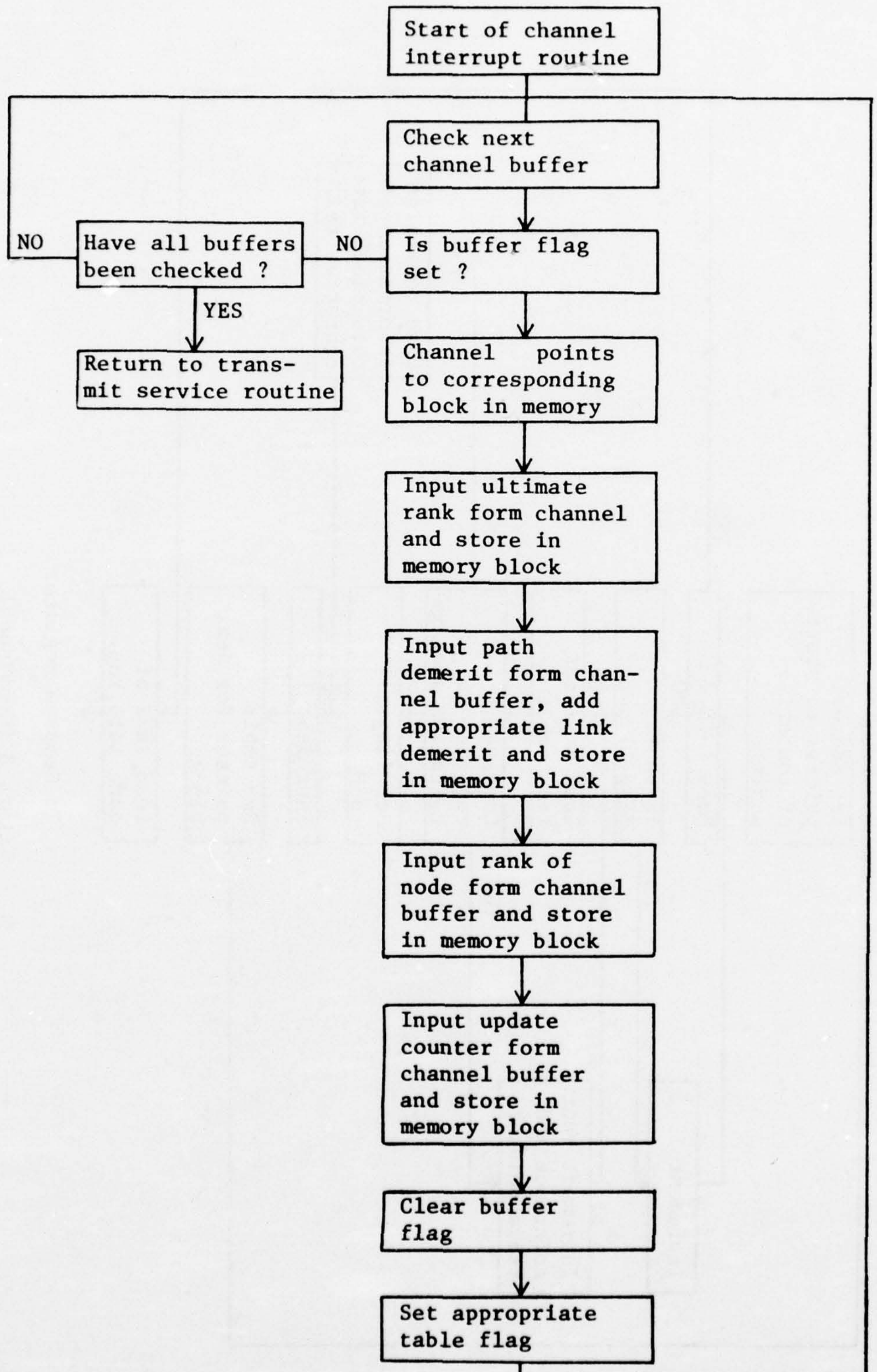
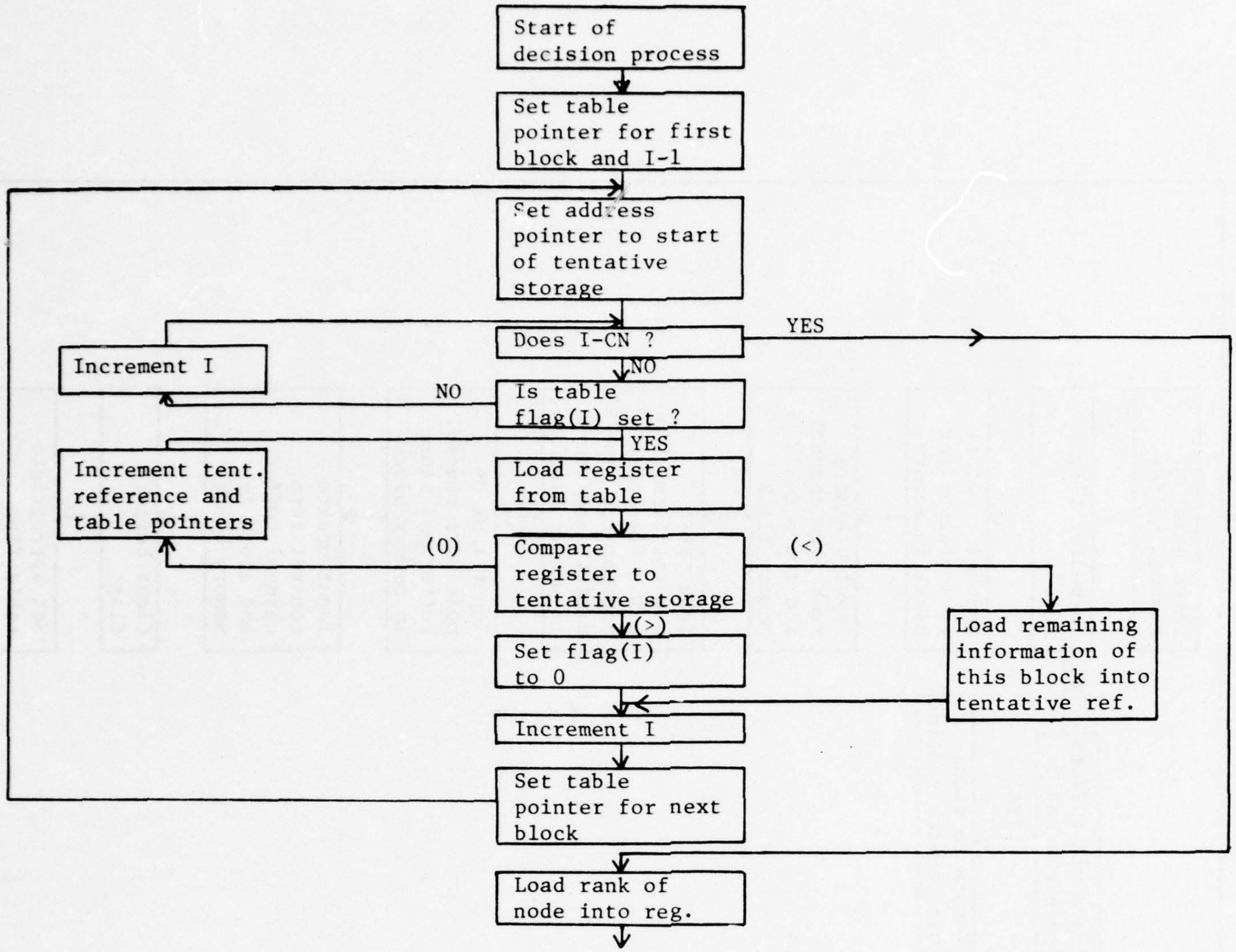


Figure 7 - Channel buffer interrupt routine for TMNB algorithm



To Compare register

Figure 8 (Continued)

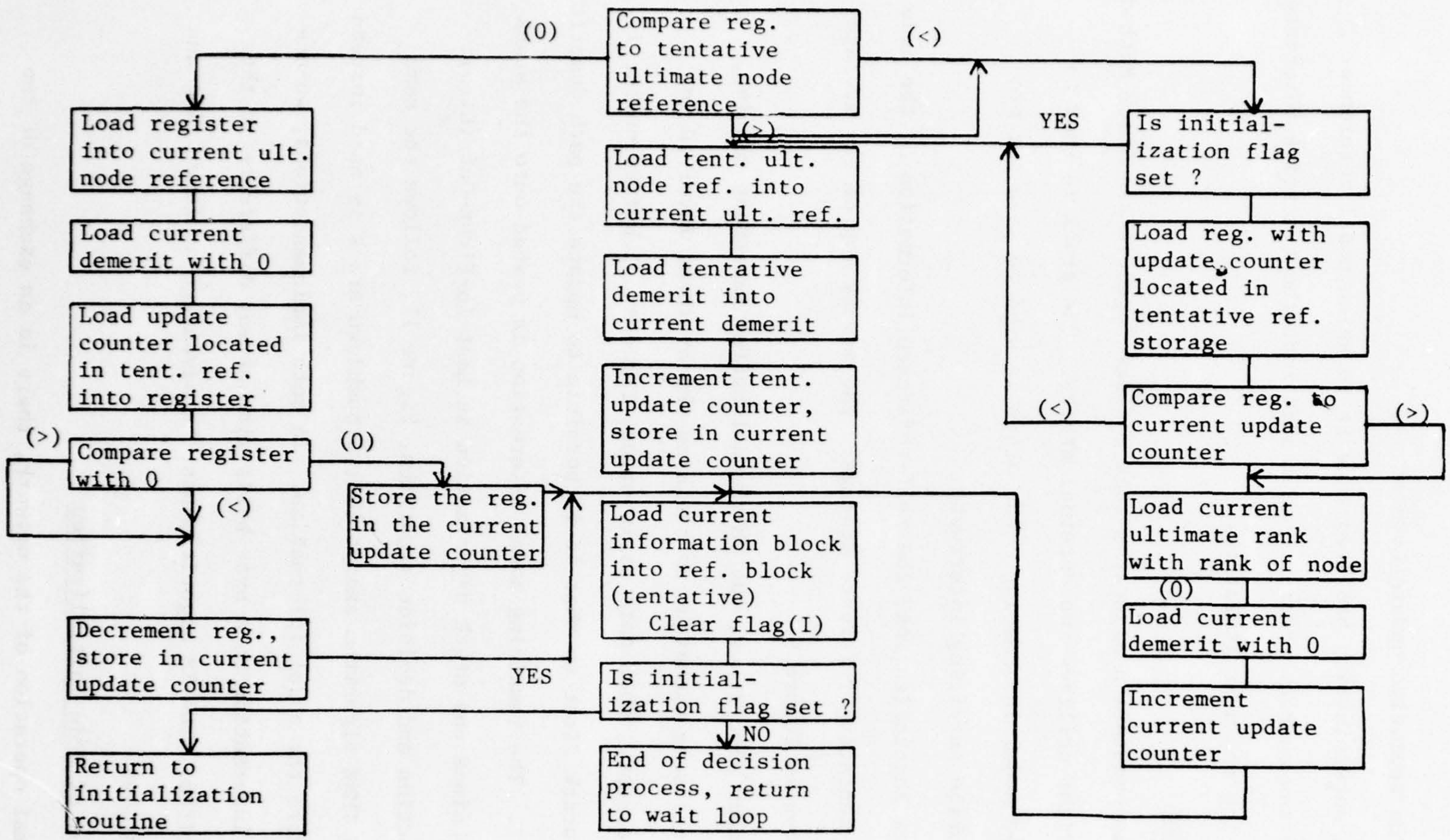


Figure 8 - Selection and decision routine for TMNB algorithm

the location of the tentative update counter.

Once a final determination has been made as to which nodal reference will be used, the new current reference is loaded into the tentative reference block for use during the next iteration.

4.3. SM Algorithm

The third implementation algorithm that was developed is the Stack Method (SM). This algorithm utilizes two external stacks. One stack is used to store the received nodal information. The other is used to save the processor's status while servicing interrupts.

During system initialization, the self-reference information of the node is stored in the tentative reference storage. The memory map of the storage allocation is shown in Figure 9.

As in the other algorithms, the SM channel buffer interrupt routine, Figure 10, uses the same technique to minimize the overhead involved in servicing an interrupt. When data is inputted, the channel buffer number is pushed onto the stack first so that it is possible to update the path demerit before storing it. The remaining nodal information is pushed onto the stack in reverse order since the stack organization is Last-In-First-Out (LIFO).

The SM selection and decision algorithm, Figure 11, follows the same procedure as the TMNB algorithm except that a pushdown stack is used instead of a table to store the nodal information. In both implementations, we are able to compare information on a byte basis without any difference in the comparison algorithm. The SM algorithm has the advantage of fewer flag and address manipulations.

4.4. The Problem of Node Initialization

During normal operation of the network, there is an exchange of two

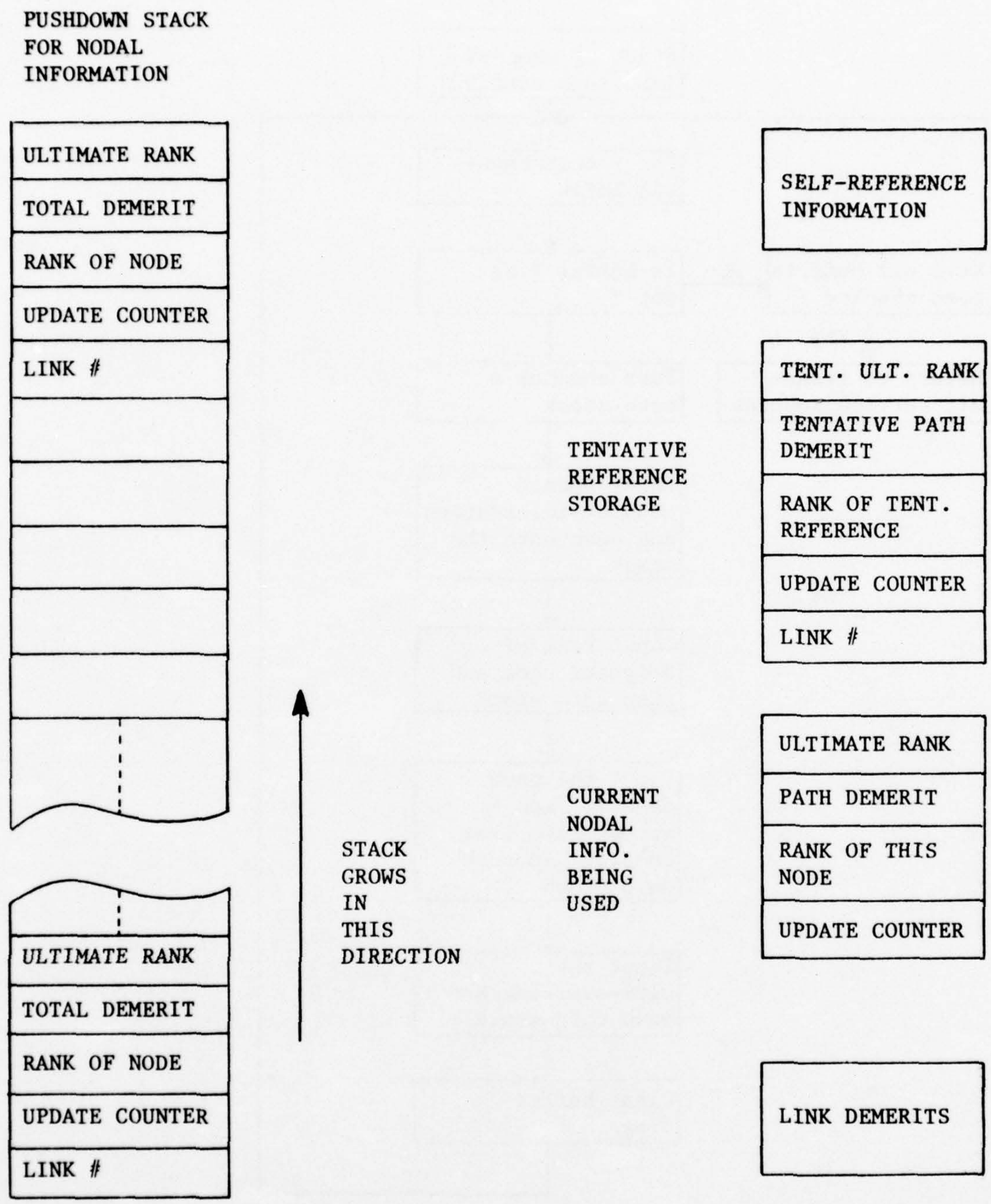


Figure 9 - Memory map for the SM algorithm.

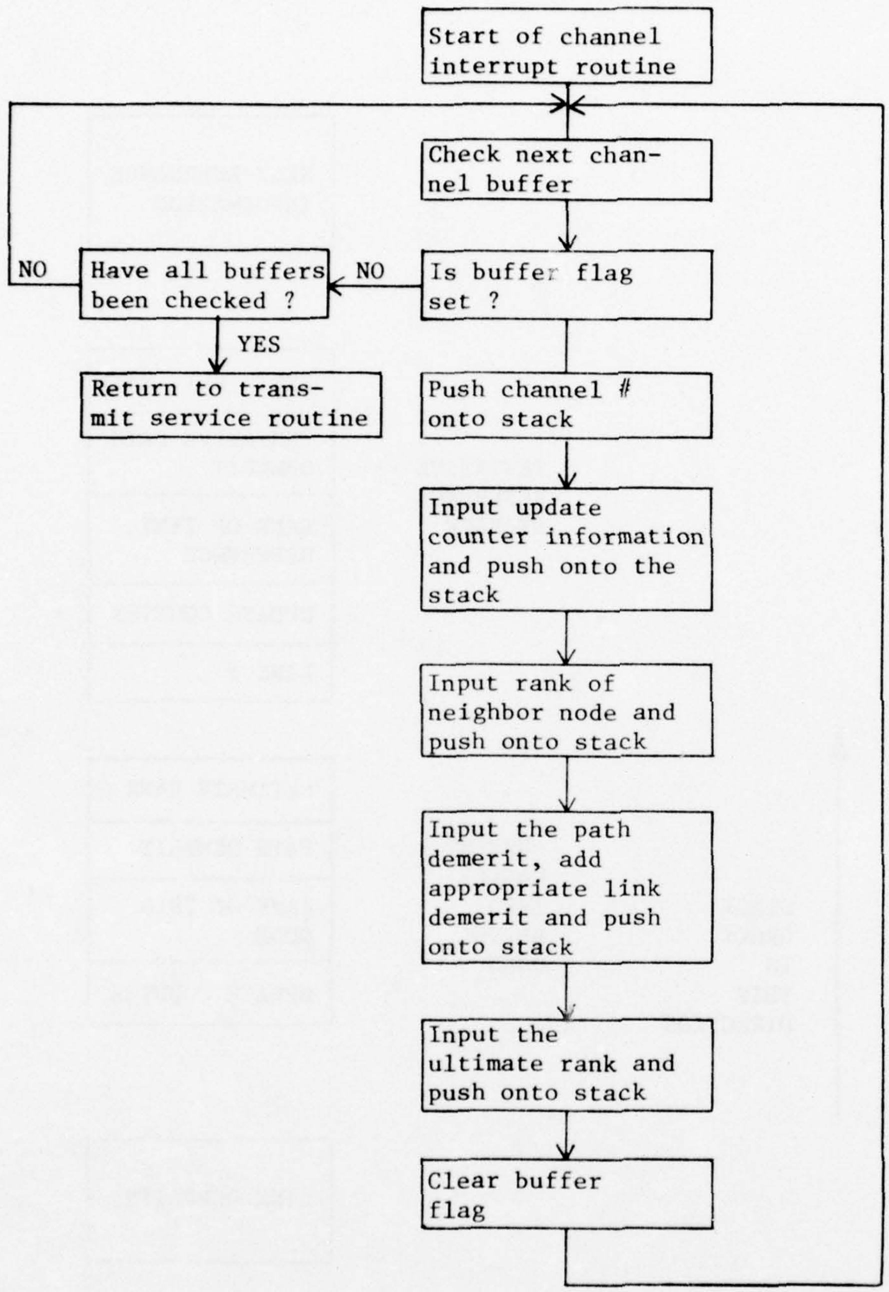


Figure 10 - Channel buffer interrupt routine for SM algorithm

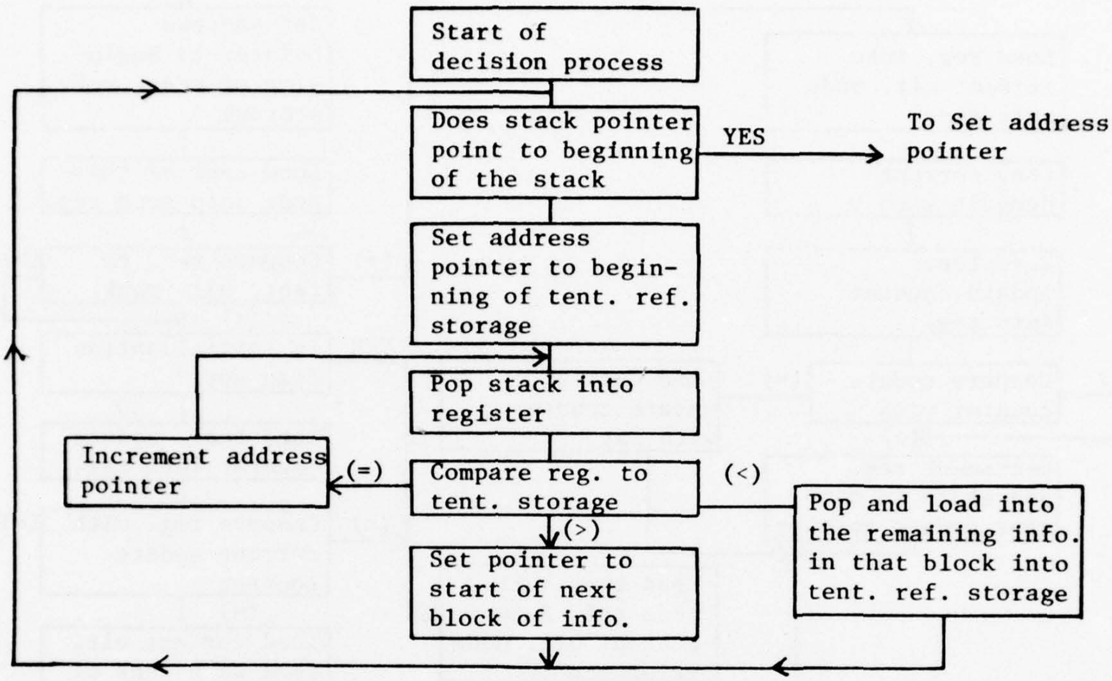


Figure 11 (Continued)

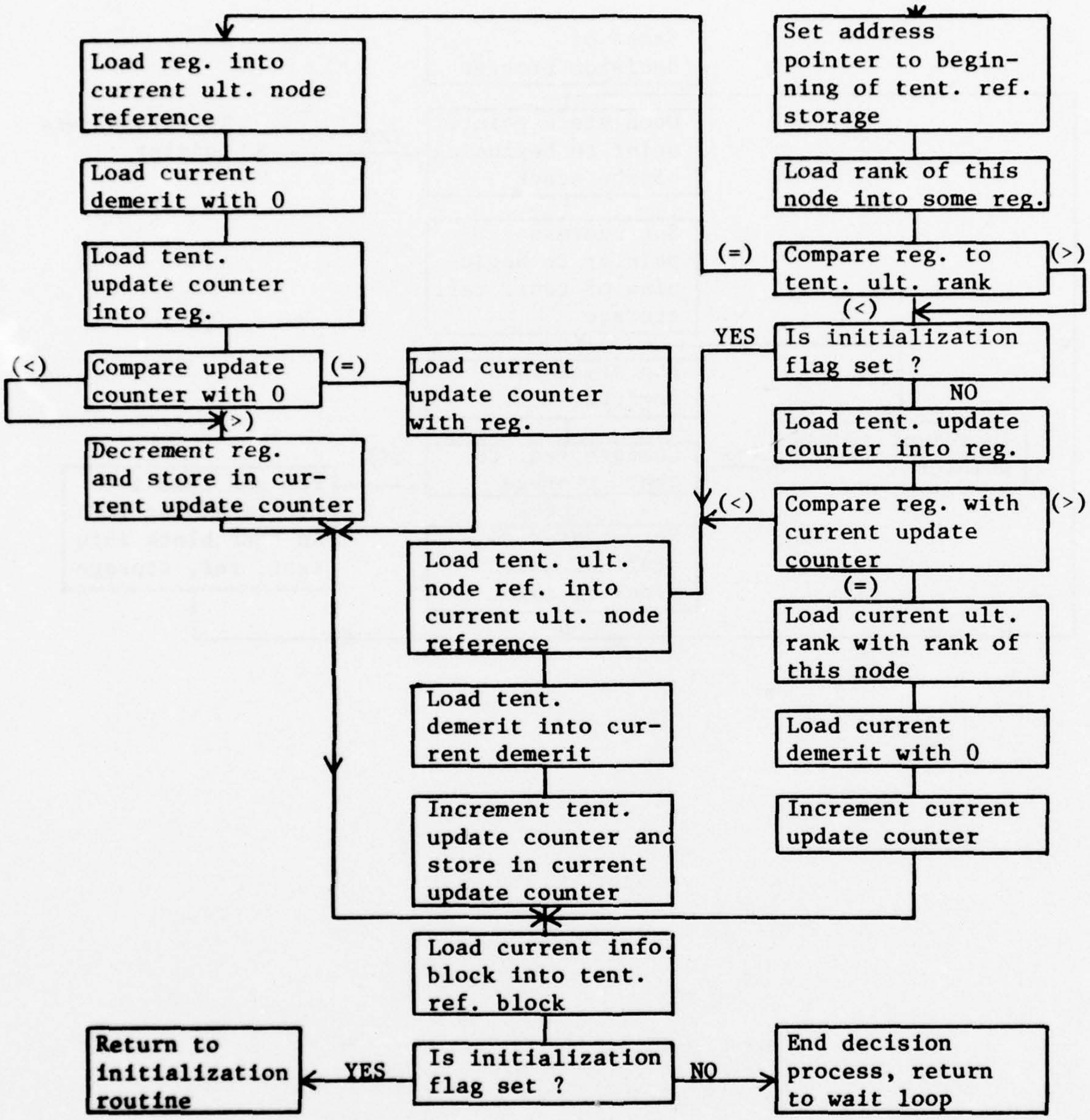
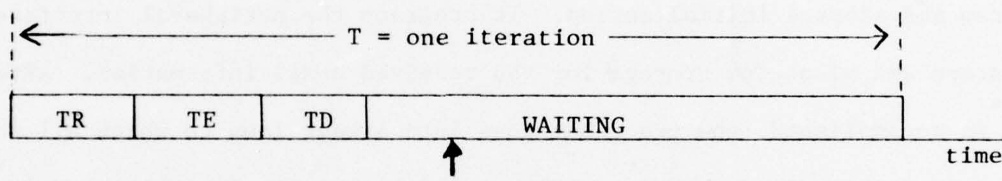


Figure 11 - Selection and decision routine for SM algorithm

kinds of information between the nodes: (i) clock time, which may be exchanged continually, and (ii) 'reference' information (used by each node to select its reference for clock time), which is exchanged at less frequent intervals. The implementation study has been concerned with the processing of the latter information, and the initialization problem arises in this regard. In the iterative scheme of reference selection, the operations at each node for one iteration can be divided into three phases followed by a waiting phase, as shown in the timing diagram below.



In the TR phase, the node transmits its reference information to its neighbors and receives similar information from them. The TE phase is a guard band between the end of the TR phase and the start of the TD phase; in the TD phase the node processes the information received in the TR phase and decides on a reference to use until the next iteration. The node then waits for the next transmission from the other nodes.

In normal operation, when all the nodes are able to refer to a common timing source, the different phases are more or less in alignment at all the nodes. Allowing for a certain guard band around the phases to take account of propagation time between nodes, the same timing diagram is then applicable to all the nodes. It is assumed here that the iteration interval T is much longer than propagation time between nodes and also much longer than (TR + TE +TD).

Suppose a new node (which knows the iteration interval T) wishes to enter the network at the instant shown by the arrow. The initialization problem is that of developing an algorithm which enables the new node to align its TR, TE and TD phases with those of the network. A more general aspect of the problem is the establishment of such alignment when all the nodes are 'new' nodes, i.e., for the initial organization of the network.

What remains to be discussed is an algorithm for the initialization of the nodal processor when that node is put into operation in the network. This algorithm, Figure 12, is organized into two parts. The first part consists of program and storage initialization. It programs the peripheral interfaces for transfers and allocates storage for the received nodal information. After this is accomplished, the processor goes into a wait loop to which all phases return during regular operation. The second part initializes the node's reference so that all nodal processors in the network initiate the same phase within a given guard band of time.

This process is accomplished by an interrupt routine which is initiated by pushing of an initialization button on an operator's console.

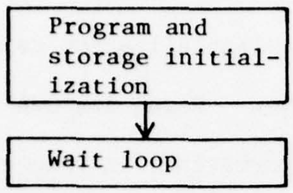
1. When the routine begins, an external counter is started and the new node starts on self-reference, sets its counter to zero and begins transmitting its timing and reference information to its neighbors. A message is included to the effect that the node is in its initialization phase.

A node disregards reference information received from nodes in their initialization phase unless

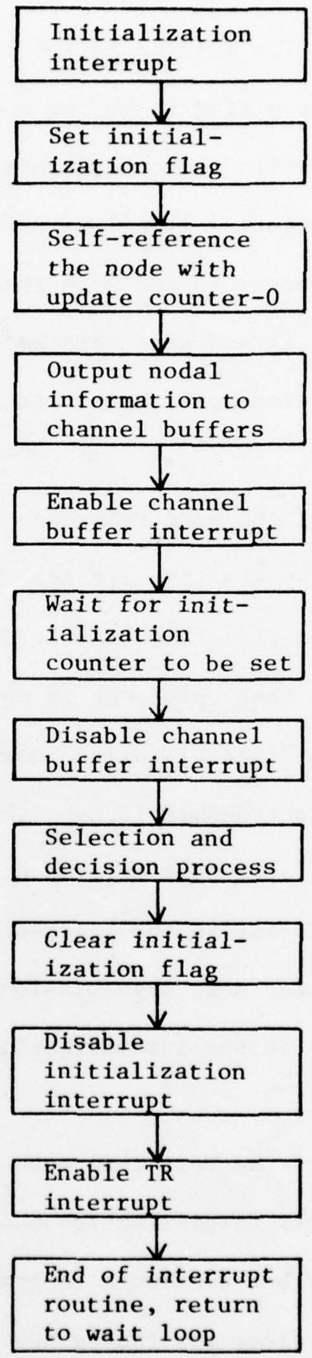
(i) the node is itself in the initialization phase, and

(ii) all the reference information it receives comes from nodes in their initialization phase.

2. The node then waits for a time T to collect information from all its



PROGRAM INITIALIZATION



NODAL REFERENCE INITIALIZATION

Figure 12 - Initialization routines

neighbors. The waiting period is controlled by the initialization counter which causes a flag to be set when the period is over. Using its own reference information and other admissible reference information, it then selects the best tentative immediate reference. If this is not self-reference, it sets its counter to one more than the largest of all the counter readings received by it and uses this best tentative reference. If the best reference is self-reference, it keeps its counter at zero and continues on self-reference.

3. (a) If the best reference is not self-reference the node initiates its iterations by starting its next TR phase at the time it receives the next transmission from this best reference.

If the best reference is not in its initialization phase, the node then cancels the "initialization" message from its transmission; otherwise, it continues to transmit it over the next iteration.

(b) If the best reference is self-reference, but at least one of its neighbors is not in the initialization phase, the node times its TR phase to begin when the next transmission is received from the highest ranking neighbor that is not in the initialization phase; it then cancels the "initialization" message.

(c) If the best reference is self-reference and all its neighbors are also in their initialization phase, the node initiates its iterations arbitrarily but continues to transmit the "initialization" message.

It is clear now that a new node discontinues its "initialization" message only when it receives information from a node which is not in the initialization phase. Once the initialization phase is over, the iterative process of transmission and reception at each node is controlled by the local clock.

The algorithm is designed to accomplish the proper initialization of new nodes joining an existing network. The same algorithm can be used when the whole network is being organized ab initio, provided one of the nodes, say, the master node, is designated as already initialized (i.e., does not transmit the "initialization" message). This node is then the "existing" network that is joined by the other nodes.

Chapter 5

A FEASIBILITY DESIGN OF NODAL PROCESSOR

In this chapter, a sample design of a nodal processor is implemented using a current state-of-the-art microprocessor. In the previous chapter, three different implementation algorithms were discussed. The TMNB and SM algorithms have a similar structure, i.e., the ultimate rank, path demerit and node rank could be concatenated together. If a microprocessor had a data path the width of the concatenation, only one comparison would have to be made on each set of received nodal information to determine the best tentative reference. However, since typical microprocessor data paths are relatively narrow, a comparison can only be made on part of the total concatenation at a time. The TMAB algorithm in contrast has a structure wherein the same type of attribute from each received node was stored in the same data block. The algorithm used flags to keep track of the comparison results. In terms of the complexity of the programming required to implement each algorithm, the TMAB algorithm is the most complex. This complexity is caused by the manipulations needed to direct the received information to their proper attribute storage block and by the flag manipulations required to control the algorithm. The TMNB algorithm ranks second in complexity. Again, the complexity is a result of the flag manipulations needed to control the algorithm. The SM algorithm is the simplest of the algorithms which were developed. The simplicity results from the use of a stack to store the received nodal information. When information is received, it is simply pushed onto the stack and when comparisons are made the information is easily popped out of the stack. In this approach, flags are not needed. The only requirement is to determine when the stack is empty. For the sample design, the implementation algorithm will be the SM algorithm.

5.1. Feasibility Design

Having decided on the algorithm, it must be shown by way of a sample design that it is feasible to implement it using a current microprocessor. The number of parameters that vary between microprocessors is large. In addition to the typical parameters that characterize all integrated circuits [9], such factors as processor architecture, system organization and software must also be investigated to determine the proper microprocessor for a particular application.

In Table 3 [22], a general comparison is made between selection criteria and their effect on the system design. For some applications, the existence of one or more features of a particular microprocessor can immediately decide the question of which microprocessor to use. In most cases, however, since several microprocessors may be comparable for an application, a different approach is needed to evaluate [23] the microprocessors. This tool is called a benchmark program. A benchmark program is a method by which the arichitecture of different processors can be evaluated by measuring their performance when executing specific functions. These functions are defined to characterize the application the processor is expected to perform. For each of the contenders, a benchmark program is written and performance parameters such as total execution time, memory space occupied and maximum I/O data rates achievable are observed and compared. Thus, the performance of competitive microprocessors can be monitored under actual operating conditions.

Finding a suitable microprocessor for this feasibility study consists of determining what specific features are needed to implement the SM algorithm.

SELECTION CRITERIA

Microprocessor Supplier

Reputation

Microprocessor availability

Documentation and application notes

Software commitment

Complete microcomputer cards or systems

Pricing and second source

Hardware

Power supply requirements

Clock and cycle time

Semiconductor technology

Interfacing requirements

Packaging

Microprocessor Architecture

Word length

Addressing capability

Registers, ALU, stack instruction set,
addressing modes and I/O

Microprocessor System

Bus control capability

Direct memory access

Interrupts

Compatible functions
- Memory
- Buffers
- Clock generators
- Input/Output
- Communications interface

Table 3 (Continued)

Microprocessor Software

Stand-alone assembler, editor, debug monitor

Cross assembler for batch and timesharing

Software documentation

Simulator

Higher level language

Table 3 (Continued)

SYSTEM DESIGN FACTORS

Is the microprocessor in production and available in quantities? The longer a product is available, the better the pricing.

Must be adequate to support design

Proves dedication to microprocessor markets

Flexibility in early hardware and software development

Cost effective with competition

Can increase total system cost

Determines speed and necessary circuit design for clock generator and timing

Is it a proven process used for other producable products?

TTL or CMOS compatibility or special

No. of packages required impacts PC board costs

Makes some applications more efficient

Enough memory for present and future

All these factors result in program efficiency, which affects amount of memory and execution speed or performance for a particular application.

Amount of external logic necessary to efficiently control peripheral devices

Needed to support high speed I/O devices

Necessary in real-time applications or for more efficient I/O

Can provide for easily assembled minimum systems or medium size systems with fewer discrete logic functions for bus control and buffering

Table 3 (Continued)

Useful in microcomputer development system during early design phases

Timeshare only feasible on small project. Batch assembler on minicomputer is very effective

Required for a good start; macro-assembler very useful

Good start up, too; food for small effort

Possible documentation and productivity benefits, but assembly language dominates

Table 3 - General comparison between selection criteria and their effect on system design.

The first necessary feature is an extensive pushdown stack to store data. An instruction set with instructions for manipulating this pushdown stack is also needed. Another important consideration is the ease of interfacing with the external channel buffers. Even more important is the capability of the microprocessor to service many sources of interrupt efficiently.

The extensive pushdown stack must be provided for externally. This can be accomplished in three different ways. The first is to have a microprocessor with built-in hardware for implementing the pushdown stack. The second is to use software to implement the stack and the third way is to implement it using microprogramming. In all, the first alternative is the easiest to use.

Looking at the currently available microprocessors, only the Intel 8080 [24] and Motorola M6800 microprocessors [25] have an internal hardware mechanism for implementing external pushdown stacks. Hence, only these two microprocessors will be considered as candidates for the design. A comparison of the capabilities of each device was made to determine which microprocessor is better suited for this application.

Both microprocessors have instructions dedicated for manipulating their external stacks. These microprocessors also have programmable peripheral interfaces which ease the interfacing problem.

The peripheral interface for the Intel 8080 has 24 I/O pins which may be programmed in two groups of twelve and can be operated in three modes. The peripheral interface adaptor (PIA) for the M6800 provides 16 bits of interface and four control lines at addressable locations in standard system memory. The I/O bits are accessed in two 8 bit words. Each bit is individually programmable as an input or output. The operating characteristics of the PIA are dynamic. These characteristics are established at system reset and can be modified at any other time by writing from the processor into registers in the PIA that control its operation.

To connect devices to the Intel 8080 bus structure, using a standard interface, requires a device called the system controller and bus driver. This device generates all necessary control signals and provides high system TTL fan-out. The M6800 however, allows up to 11 devices to be directly connected to its bus structure before buffering is needed. A fundamental characteristic of the bus structure is that the processor references all bus components as memory locations. All connected devices have the same interface as memory and no I/O control lines are needed on the bus.

In Chapter 2, the interrupt structures of both microprocessors were discussed.

After looking at the various capabilities of both microprocessors, the M6800 was better suited for the sample design. Although both processors have similar capabilities, by looking at the result of the benchmark programs in Chapter 2, it is evident that the M6800 is genrally more efficient. This shows that even though both processors have the same instruction cycle, the instructions of the M6800 are more powerful in what they can accomplish.

Figure 13 is a block diagram of a nodal processor design using the M6800 microprocessor. The processor is controlled by interrupts generated by external harware and PIA's which are the interface between the microprocessor and the channel buffers. What is not shown in the block diagram is the buffering needed if more than 11 devices are connected to the busses. Bus drivers and receivers are available which allow up to 50 receivers for

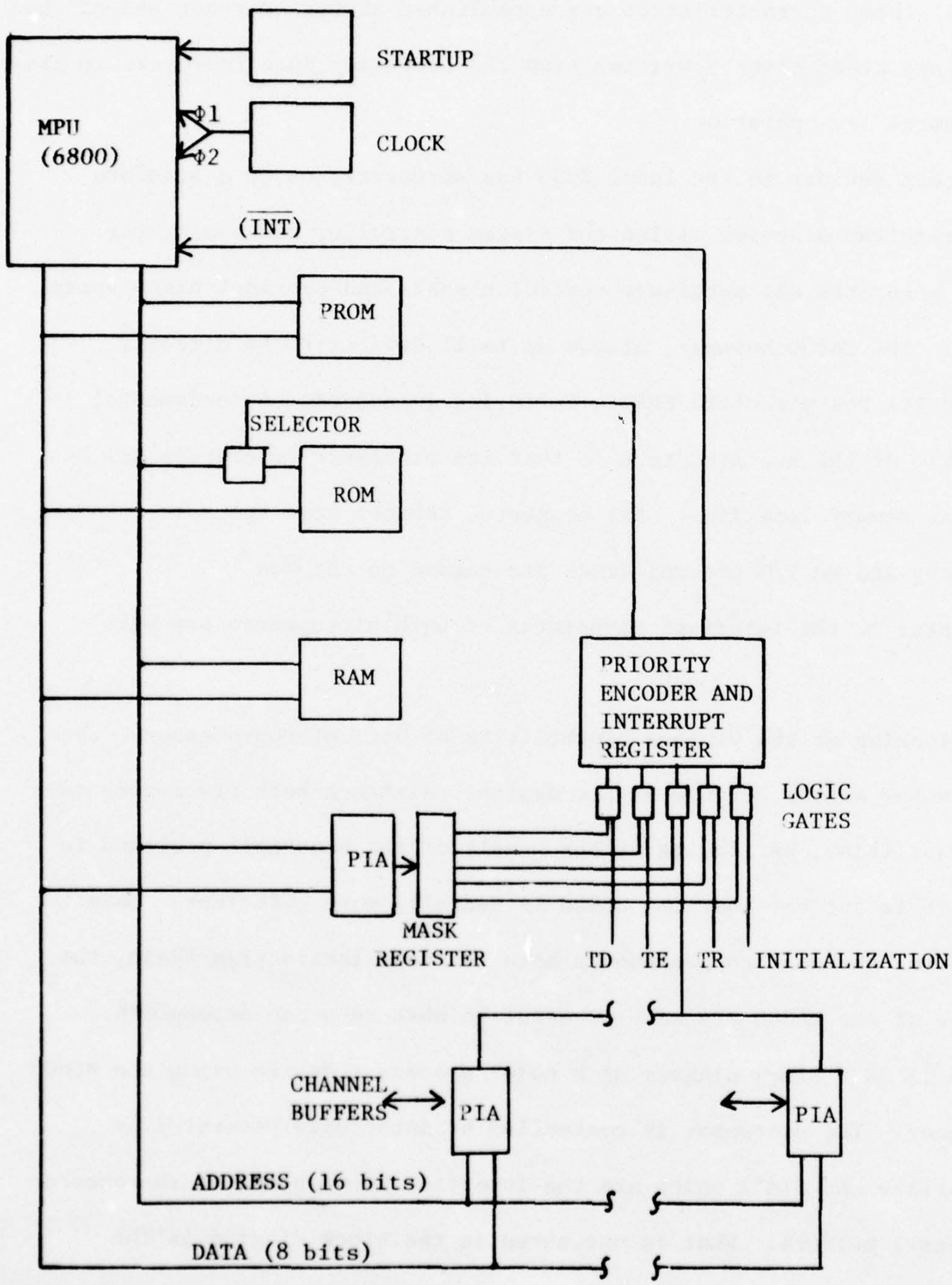


Figure 13 - Block diagram of nodal processor.

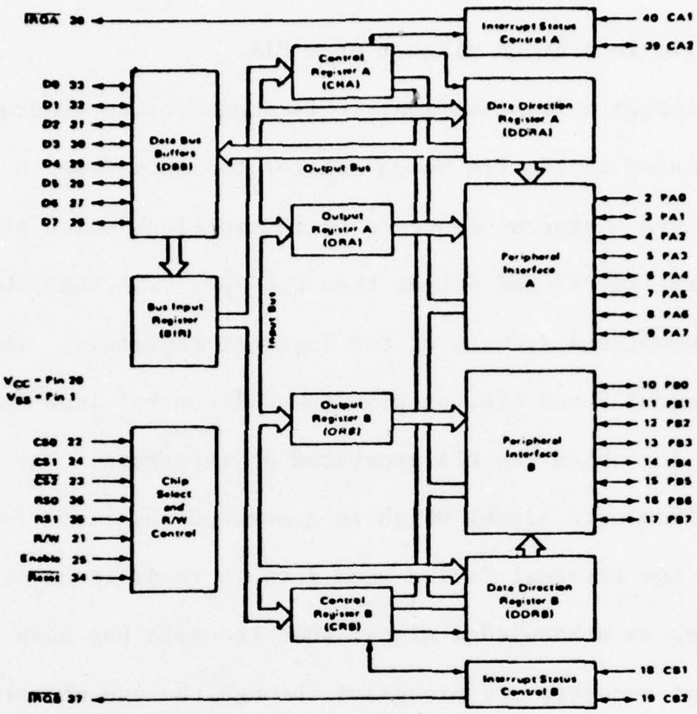
each driver.

Figure 14a is a block diagram of a PIA.

During system initialization, it is necessary to program the control registers located in the PIA which control the data flow thru the interface. Figure 14b gives a general sequence of instructions which programs the PIA for input thru PA0-PA7 and output thru PB0-PB7. CRA, CRB, DRA and DRB refer to general symbolic addresses of the internal registers. The specific hexadecimal numbers $2F and $24, program the CA1 control line to act as a data ready signal for which the PIA generates an interrupt. The CA2 line acts as a data acknowledge signal which is generated when data is input. The CB1 line signals the external device when data is ready to be output. The CB2 line generates an acknowledge signal when the data has been accepted.

Interrupt requests are processed through the use of external priority hardware and internal software polling. In addition, a programmable mask register is provided which can disable each source of interrupt individually. When a normal interrupt request occurs, the microprocessor addresses locations FFF8 and FFF9 and loads the contents into the program counter. The program counter then contains either the starting address of the interrupt routine or the address of a sequence of registers which are used for software polling. External priority hardware causes each interrupt received to go to a different trap location in memory. Each trap location contains a vector to the specific interrupt routine. All the interrupts are priority encoded. The channel buffer interrupt also includes a software polling sequence that determines which channel buffer is to be serviced.

Appendix A contains programs coded for the M6800 microprocessor. All three phases of the SM algorithm are presented. These programs <u>should not</u>



(a)

```
CLR A
STA A  CRA
STA A  CRB
STA A  DRA        SET PA0-PA7 FOR INPUTS
LDA A  #$2F
STA A  CRA        SET CONTROLS
LDA A  #$FF
STA A  DRB        SET PB0-PB7 FOR OUTPUTS
LDA A  #$24
STA A  CRB        SET CONTROLS
```

(b)

Figure 14 - (a) Block diagram of a PIA, (b) PIA initialization program

be considered as an implementation since they have not been assembled to check for coding errors or simulated to check for errors in the programming of the algorithm. However, they are sufficient as a means of establishing a reasonable timing and storage estimate. The timing information consists of an estimate of the time needed to complete each phase on a worst case basis.

Two basic assumptions were made for all of the routines. First, all direct addressing was in the extended mode. Second, a data acknowledge signal from a device is always generated within 4 μs.

5.2. Estimates of Processing Time

The time required for the transmit and receive phase consists of the time required for the TR routine plus the time needed for inputting nodal information from N channel buffers. An assumption made with regard to the channel buffer routine was that only one buffer was serviced per interrupt request. Equation (1a) determines the time needed to complete the TR routine as a function of the N nodes to which it is connected. The unit of time is the microsecond.

$$t_{TR} = N(417) + 71 \ . \qquad (1a)$$

Equation (1b) determines the time needed to service the channel buffer routine N times.

$$t_{CHNBUF} = N^2(54) + N(558) \qquad (1b)$$

Equation (1c) is the time required to complete the transmit and receive phase for N connected nodes.

$$T_1 = t_{TR} + t_{CHNBUF} \ . \qquad (1c)$$

The time needed for the second phase consists of the time required to complete the TE routine. Equation (2) represents this time as a function of N connected nodes.

$$T_2 = N(25) + 42 . \tag{2}$$

The time requirement for the third phase consists of the time needed to complete the selection and decision routine for N connected nodes. An assumption is made that the first 5 bytes of the information received from each node are the same. Also the first byte of the rank of each node is equal. Since all ranks are unique, the time estimate equation of the third phase will only be valid for a maximum of 256 connected nodes. During the selection process, two different cases arise in determining the worse case time. The first case is that the current reference is better than each of the N connected nodes. It is given by equation (3a).

$$t_{SP_1} = N(192) + 14 . \tag{3a}$$

The second case is while sequentially testing the nodes, the information of the next node is better than the previous node. The time required is given by equation (3b).

$$t_{SP_2} = N(252) + 14 . \tag{3b}$$

Clearly, equation (3b) is the worst case for the selection process.

During the decision process three different worst case possibilities exist. Case 1 consists of the node self-referencing itself since the tentative ultimate rank is equal to the rank of this node. The time required is given by equation (3c).

$$t_{DP_1} = 230 . \tag{3c}$$

For Case 2, the tentative and current update counters are compared, with the tentative update counter being greater than the current counter. The time required for this case is given by equation (3d).

$$t_{DP_2} = 280 \ . \tag{3d}$$

Case 2 is the situation wherein the current counter is greater than the tentative update counter. The time required in this case is given by equation (3e).

$$t_{DP_3} = 321 \ . \tag{3e}$$

Clearly, Case 3 is the worst case. Therefore, the time required to complete the worst case third phase is given by equation (3f).

$$T_3 = N(252) + 235 \ . \tag{3f}$$

Table 4 gives a time estimate for completing each phase of the iteration process for two different numbers of nodes that are connected to a single node. These estimates give an approximate upper bound on the time needed to complete an iteration.

TIME PER PHASE	NUMBER OF CONNECTED NODES (N)	
	N = 10	N = 100
T_1	14621 μs.	631571 μs.
T_2	292 μs.	2542 μs.
T_3	2855 μs.	25535 μs.
TOTAL	17.8 ms.	.66 s.

Table 4. Phase completion time estimates

5.3. Estimates of Needed Storage

An estimate of storage needed, both ROM and RAM, as a function of the number of connected nodes is an essential parameter for system design. The storage requirements for the program initialization, nodal reference initialization, TR, channel buffer, TE and TD interrupt routines are not dependent on the number of connected nodes N. However, the number of link demerits and the storage configuration number are dependent on the number of connected nodes.

Since the interrupt routines are the same for each node, an identical ROM could be masked. However, since the link demerits, storage configuration number and rank of the node are different for each node, a programmable ROM should be used. An estimate of the required PROM storage for this data is given by equation (4).

$$B_1 = N + 5 . \tag{4}$$

The estimated bytes of storage needed for the interrupt routines plus constants is given by equation (5).

$$B_2 = 608 . \tag{5}$$

The byte requirements for RAM storage must include space for the variables, the current and tentative nodal information, a stack for subroutines and interrupt linkage and a data stack which is a function of the connected nodes. This estimate is represented by equation (6).

$$B_3 = 10N + 38 . \tag{6}$$

Table 5 is an estimate for the amount of each type of storage required. PROM storage is represented by B_1, ROM storage by B_2 and RAM storage by B_3.

TYPE OF STORAGE	NUMBER OF CONNECTED NODES	
	N = 10	N = 100
B_1 (PROM)	15 bytes	105 bytes
B_2 (ROM)	608 bytes	608 bytes
B_3 (RAM)	138 bytes	1038 bytes

Table 5. Storage estimates for routines

When a new node is brought into a network, it is necessary to make physical changes to each of the nodes that will be connected to the new node. These changes include increasing the number of channel buffers and RAM storage and also the modification of the data that is stored in the PROM. To provide a node with the capability of making these changes without disrupting its operation, each node could have two nodal processors. One processor will be on-line at all times. If any modifications are necessary, they are done on the processor that is off-line. To bring the modified processor on-line, the following procedure is followed. The modified processor waits until the on-line processor has finished the TD interrupt routine. It then initializes its status to the current status of the on-line processor. Once initialized, the processor then switches itself on-line and the other processor off-line.

An alternative approach to this problem is to shut down the processor to make any required changes. During the period when changes are being implemented, the timing at a given node is allowed to run freely. Since the local clocks at a node should be highly stable, they should be able to maintain timing in a flywheel mode for a sufficient period of time. Once the required changes have been made to the nodal processor, its operation is resumed by initiating the initialization procedure that was previously described.

Chapter 6

CONCLUSION

In essence, this study has demonstrated the feasibility of using a microprocessor to dynamically allocate information paths in a Time Reference Distribution (TRD) system. The microprocessor approach was investigated since it represented the best design tradeoffs from the reliability, versatility and economic viewpoints.

Three basic organizations were developed for implementing a TRD nodal control processor. A comparative evaluation of these approaches indicated that, although all these techniques are feasible, the stack oriented organization presented superior design alternatives. This design was then pseudo-encoded for a currently available microprocessor to determine the PROM, ROM, and RAM memory requirements, since these usually represent the dominant cost and reliability factors. Execution time estimates were also generated from this encoding. The sample design has shown that the addition of new nodes into the network does not require extensive redesign of the nodal processor.

Once initial design costs have been paid, a microprocessor nodal system to handle up to one hundred interconnecting nodes can be constructed for under $100.00 based on todays prices for microprocessors, memories and interface devices. Even at this low price, the system would be easily reprogrammable by simply changing the ROM or PROM chips. Once past infancy failure, the integrated circuit chips, themselves, are infinitely reliable in comparison to the failure of interconnections. However, a system composed of a microprocessor, 1024 words of RAM, 1024 words of ROM, a programmable 15 bit interface, and 256 words of PROM consists of only seven integrated circuits and one power supply and hence, represents a rather minimal number of interconnections.

Finally, when one considers other functions which must be performed at a communications node, the microprocessor approach becomes even more attractive. For example, switching, monitoring, etc. all readily lend themselves to microprocessor implementations. Hence, the overall life cycle costs can be reduced and the reliability increased by incorporating many nodal functions into one microprocessor system.

The following areas are suggested for investigation in future studies of the dynamic allocation process:

1) A hardware simulator should be built with an interconnection to a minicomputer. This would allow the algorithms that were developed to be evaluated under controlled conditions in a simulated network environment.

2) For a digital communications network, useful measures for path demerits and nodal ranks should be developed. Using these measures, methods for making automatic time corrections should be studied.

REFERENCES

[1] Special issue on Time and Frequency, Proc. IEEE, Vol. 60, No. 5, May 1972.

[2] H. C. Folts, "Time and Frequency for Digital Communications," Proc. of the Fourth Precise Time and Time Interval Planning Conference, (NASA and Department of Defense) pp. 194-202, November 1972.

[3] H. W. Hellwig, "Atomic Frequency Standards: A Survey," Proc. IEEE, Vol. 63, No. 2, pp. 212-229, February 1975.

[4] J. L. Jesperson, B. E. Blair and L. E. Gatterer, "Characterization and Concepts of Time-Frequency Dissemination," Proc. IEEE, Vol. 60, No. 5, pp. 502-521, May 1972.

[5] G. P. Darwin and R. C. Prim, "Synchronization in a System of Interconnected Units," U.S. Patent 2986723, May 1961.

[6] H. A. Stover, "Coordinated Universal Time (UTC) as a Timing Basis for Digital Communications Networks," Eascon "74 Record, pp. 649-654, October 1974.

[7] J. W. Pan, "Synchronizing and Multiplexing in a Digital Communications Network," Proc. of IEEE, Vol. 60, No. 5, pp. 594-601, May 1972.

[8] D. A. Perreault, R. Mukundan and K. R. Krishnan, "Dynamic Allocation of Hierarchical Information Paths for the Organization of Interconnected Systems," Proc. 1976 IEEE International Symposium on Circuits and Systems, pp. 390-393, Munich, April 1976.

[9] The TTL Data Book for Design Engineers, Texas Instruments, Inc.

[10] S. Runyon, "Microprocessors Showing Promise in Test Equipment, But Haven't Made It Big Yet," Electronic Design 9, April 26, 1974.

[11] B. Gladstone, "Designing with Microprocessors Instead of Wired Logic Asks More of Designers," Electronics, October 11, 1973.

[12] "Debugging Microprogrammed Systems," Application Notes, Scientific Micro Systems, Inc.

[13] G. W. Schultz, R. M. Holt and H. L. McFarland, "A Guide to Using LSI Microprocessors," Computer, Vol. 6, No. 6, June 1973.

[14] W. Davidow, "How Microprocessors Boost Profits," Electronics, July 11 1974.

[15] D. J. Theis, "Microprocessor and Microprocessor Survey," Datamation, December 1974.

[16] L. Altman, "The New LSI," Electronics, July 10, 1975.

[17] "Comparison of MCP1600 with Presently Available Microprocessors," Western Digital Corporation.

[18] A. G. Vacroux, "Explore Microcomputer I/O Capabilities," Electronic Design 10, May 10, 1975.

[19] S. S. Husson, Microprogramming Principles and Practice, Prentice Hall, Inc., 1970.

[20] D. Knuth, The Art of Computer Programming Vol. 1, Addison-Wesley, 1968.

[21] T. Blakeslee, Digital Design with Standard MSI and LSI, J. Wiley and Sons, Inc., 1975.

[22] A. Lane, "Microprocessor-System Design," Digital Design, Autust 1975.

[23] M. Lewin, "Integrated Microprocessors" Trans. IEEE Circuits and Systems, Vol. CAS-22, pp. 577-585, July 1975.

[24] "Intel 8080 Microcomputer System Manual," Intel Corporation, January 1975.

[25] "M6800 Microprocessor Applications Manual," Motorola Semiconductor, 1975.

APPENDIX A

PROGRAMMING LISTINGS

```
*                                                        *
*                   TR INTERRUPT                         *
*                                                        *
*     INTERRUPT ROUTINE FOR THE TRANSMISSION OF THE NODE'S *
*     CURRENT REFERENCE TO EACH OF ITS OUTPUT CHANNEL    *
*     BUFFERS.                                           *
*                                                        *
TR      STS    STK        STORE STACK POINTER
        CLR    OFFSET
        CLR    OFFSET+1
NXBUF   LDX    #0         INITIALIZE INDEX REGISTER
        LDA B  #8         LOAD WORD COUNT
OUT     LDA A  CURANK,X   INPJT BYTE FROM CURRENT INFO BLOCK
        INX
        TXS               SAVE INDEX REG IN STACK POINTER REG
        LDX    OFFSET
        STA A  ORB,X      STORE BYTE IN OUTPJT BUFFER
TEST    TST    CRB,X      WAS BYTE ACCEPTED?
        BPL    TEST       IF NO,TEST AGAIN
        DEC B             DECREMENT WORD COUNT
        TSX               RELOAD INDEX REG
        TST B             IS TRANSFER COMPLETE
        BNE    OUT        IF NO, TRANSFER NEXT BYTE
        LDA A  OFFSET+1    IF YES, SET UP POINTER
        ADD A  #2         TO NEXT CHANNEL BUFFER
        STA A  OFFSET+1
        LDA A  OFFSET
        ADC A  #0
        STA A  OFFSET
        LDX    CONFIG     LOAD CONFIG. NUMBER
        CPX    OFFSET     HAVE ALL CHANNEL BUFFERS BEEN LOADED?
        BGT    NXBUF      IF NO, OUTPUT TO NEXT BUFFER
        LDS    STK        RELOAD STACK POINTER
        LDX    #NODES
        STX    PRSTSP
        LDA A  ETECB      LOAD INTERRUPT MASK
        STA A  MSKDR      STORE IN MASK REG
MASK1   TST    MSCRB      TRANSFER COMPLETE?
        BPL    MASK1
        TST    INTFLG     IS INITIALIZATION FLAG SET?
        BPL    INTR
        RTS    IF YES, RETURN TO INITIAL ROUTINE
INTR    RTI               IF NO, RETURN AND WAIT FOR CHANNEL
                          BUFFER AND TE INTERRUPTS.
```

```
*                                                               *
*                  CHANNEL BUFFER INTERRUPT                     *
*                                                               *
*       INTERRUPT ROUTINE TO INPUT INFORMATION FROM CHANNEL     *
*       BUFFERS. THERE IS A CHANNEL BUFFER FOR EACH DIRECTLY    *
*       CONNECTED NODE.                                         *
*                                                               *
CHNBUF  CLR       OFFSET
        CLR       OFFSET+1
        STS       STK       STORE PRESENT STACK POINTER
        LDS       PRSTSP
NXTCHL  LDA B     #10
        LDX       OFFSET    LOAD WORD COUNT
        TST       CRA,X     TEST FLAG OF INPUT BUFFER
        BPL       CHNNL     IF NOT SET, CHECK NEXT BUFFER.
INPUT   LDA A     ORA,X     IF SET, INPUT BYTE
        CMP B     #10
        BNE       B10
        STA A     LNKPNT
B10     CMP B     #9
        BNE       B9
        STA A     LNKPNT+1
B9      PSH A               PUSH ONTO STACK
        DEC B               DECREMENT WORD COUNT
        BEQ       CHNNL     IF WORD COUNT IS 0, CHECK NEXT BUFFER
LOOP    TST       CRA,X     DETERMINE IF NEXT BYTE IS READY
        BPL       LOOP
        CMP B     #4        IS NEXT BYTE,M.S. BYTE OF DEMERIT?
        BEQ       PTHDEM    IF YES, BRANCH TO PTHDEM
        CMP B     #3        IS NEXT BYTE,L.S. BYTE OF DEMERIT?
        BEQ       ADDLNK    IF YES, BRANCH TO ADDLNK
        JMP       INPUT
PTHDEM  LDA A     ORA,X     LOAD M.S. BYTE OF DEMERIT INTO A REG
        CEC B
        JMP LOOP
ADDLNK  DEC B
        STA B     WRDCNT
        LDA B     ORA,X     LOAD L.S. BYTE OF DEMERIT INTO B REG
        STX       INDX
        LDX       LNKPNT
        ADD B     LINK,X    ADD LINK DEMERIT TO PATH DEMERIT
        PSH A               LOAD UPDATED DEMERIT ONTO STACK
        PSH B
        LDX       INDX
        LDA B     WRDCNT
        JMP       LOOP
CHNNL   LDA A     OFFSET+1 DETERMINE THE ADDR OF NEXT INPUT BUF.
        ADD A     #2
        STA A     OFFSET+1
        LDA A     OFFSET
        ADC A     #0
        STA A     OFFSET
```

```
LDX     CONFIG   HAS ALL INPUT BUFFERS BEEN CHECKED?
CPX     OFFSET
BGT     NXTCHL   IF NO, CHECK NEXT BUFFER
STS     PRSTSP
LDS     STK      IF YES, RETURN FROM ROUTINE AND WAIT
RTI              FOR NEXT INTERRUPT
```

```
*                                                                    *
*                        TE INTERRUPT                                *
*                                                                    *
*      TRANSMIT END INTERRUPT ROUTINE - CLEARS ALL CHANNEL           *
*      INPUT BUFFER FLAGS AND DISABLES CHANNEL BUFFER                *
*      INTERRUPTS.                                                   *
*
TE      CLR     OFFSET
        CLR     OFFSET+1
        LDX     OFFSET
CLEAR   STA A   ORA,X     CLEAR INTERRUPT FLAGS FOR INPUT
        CPX     CONFIG    CHANNEL BUFFERS. HAVE ALL FLAGS BEEN
        REQ     RETURN    CLEARED?
        INX
        INX
        JMP CLEAR
        TST     INTFLG    IS INITIALIZATION FLAG SET?
        BMI     EXIT
RETURN  LDA A   ETD
        STA A   MSKDR     LOAD MASK REG
MASK2   TST     MSCRB
        BPL     MASK2
        RTI
EXIT    LDA A   DCB       DISABLE CHANNEL BUFFER INTERRUPTS
        STA A   MSKDR
MASK3   TST     MSCRB
        BPL     MASK3
        RTS
```

```
*
*                   TD INTERRUPT
*
*    THE SELECTION AND DECISION INTERRUPT ROUTINE IS        *
*    USED TO DETERMINE THE BEST MODAL REF. TO BE USED       *
*    DURING THE NEXT ITERATION TIME INTERVAL. THE ROUTINE   *
*    USED MODAL INFORMATION THAT WAS STORED IN A STACK      *
*    DURING THE CHANNEL BUFFER INTERRUPT ROUTINE            *
*
TD     STS    STK       STORE INTERRUPT SP
       LDS    PRSTSP    LOAD SP WITH ADDR OF STORAGE STACK
LOOP1  LDX    #TENT     LOAD INDEX REG-START ADDR OF TENT. REF
NEXT   PUL A            LOAD A FROM STACK
       BEQ    CASE1
       BGT    CASE2
LOOP2  CPX    #TENT+10  TEST FOR END OF BLOCK
       BEQ    EMPTY
       STA A            STORE INFO OF THAT BLOCK INTO TENT. REF
       INX
       PUL A
       JMP    LOOP2
CASE2  CPX    #TENT+10 SET SP TO START OF NEXT NODE'S INFO
       BEQ    EMPTY
       INX
       INS    INCREMENT SP
       JMP    CASE2
CASE1  INX              INCR. ADDR POINTER
       JMP    NEXT
EMPTY  LDX    #NODES    IS STORAGE STACK EMPTY?
       STS    PRSTSP
       CPX    PRSTSP
       BLE    DECISN
       JMP    LOOP1
DECISN LDX    #TENT     LOAD INDEX REG - START OF TENT. REF
       LDA A  SLFREF
       LDA B  SLFREF+1  DETERMINE IF BEST TENT REF IS SELF-REF
       CMP A  X
       BNE    NSREF
       INX
       CMP B  X
       BEQ    SREF
NSREF  TST    INTFLG
       BMI    NEWCUR    BRANCH IF INITIAL. FLAG IS SET
       LDX    #TENT+6
       LDA A  X         LOAD IN TENT UPDATE COUNTER
       INX
       LDA B  X
       CMP A  CCOUNT    COMPARE M.S. BYTE OF CURRENT COUNTER
       BLT    NEWCUR
       BGT    TGTC
       CMP B  CCOUNT+1 L.S. BYTE
       BLT    NEWCUR
```

```
TGTC    LDA A  SLFREF
        LDA B  SLFREF+1
        STA A  CURANK    LOAD CURRENT ULT. RANK WITH RANK OF
        STA B  CURANK+1  THIS NODE
        CLR    CDEMRT    LOAD CURRENT DEMERIT WITH 0
        CLR    CDEMRT+1
        LDX    CCOUNT
        INX
        STX    CCOUNT    INCR. CURRENT UPDATE COUNTER
        JMP    LDTENT
NEWCUR  LDX    #TENT
        LDA A  X
        INX
        STA    CURANK    LOAD TENT ULT NODE REF INTO CURRENT
        LDA A  X         ULT REF
        INX
        STA A  CURANK+1
        LDA A  X
        INX
        STA A  CDEMRT
        LDA A  X
        INX
        STA A  CDEMRT+1
        LDA A  #1
        ADD A  X+1       INCR. TENT UPDATE COUNTER
        CLR B
        ADC B  X
        STA A  CCOUNT+1  STORE IN CURRENT UPDATE COUNTER
        STA B  CCOUNT
        JMP    LDTENT
SREF    STA A  CURANK    STORE RANK OF THIS NODE IN CURRENT
        STA B  CURANK+1  ULT NODE REF
        CLR    CDEMRT    SET THE CURRENT DEMERIT TO 0
        CLR    CDEMRT+1
        LDX    TENT+6    LOAD TENT UPDATE COUNTER INTO INDEX REG
        CPX    #0        COMPARE TO 0
        BEQ    STORE     IF=0, BRANCH TO STORE
        DEX              OTHERWISE, DECREMENT
STORE   STX    CCOUNT    STORE IN CURRENT UPDATE COUNTER
LDTENT  LDX    #TENT
        LDA A  CURANK    STORE CURRENT INFO BLOCK INTO TENT
        LDA B  CURANK+1  REF BLOCK
        STA A  X
        INX
        STA B  X
        INX
        LDA A  CDEMRT
        LDA B  CDEMRT+1
        STA A  X
        INX
        STA B  X
        INX
```

```
        LDA A   SLFRFF
        LDA B   SLFREF+1
        STA A   X
        INX
        STA B   X
        INX
        LDA A   CCOUNT
        LDA B   CCOUNT+1
        STA A   X
        INX
        STA B   X
        LDS     STK
        LDA A   CTR
        STA A   MSKDR     LOAD INTERRUPT MASK REG
MASK4   TST     MSCRB
        BPL     MASK4
        TST     INTFLG    IS INITIALIZATION FLAG SET?
        BMI     INT
        RTI               IF NO, RETURN FROM INTERRUPT AND WAIT
INT     RTS               IF YES, RETURN TO INITIALIZATION ROUTINE
```

```
*                                                           *
*               INITIALIZATION INTERRUPT                    *
*                                                           *
*     THIS INTERRUPT ROUTINE INITIALIZES THE NODE'S         *
*     REFERENCE IN THE NETWORK                              *
*                                                           *
INTIAL LDA A   #1
       STA A   INTFLG    SET INITIALIZATION FLAG
       LDA A   SLFREF    SET NODE ON SELF-REFERR
       LDA A   SLFREF    SET NODE ON SELF-REFERENCE
       LDA B   SLFREF+1
       STA A   CURANK
       STA B   CURANK+1
       CLR     CDEMRT
       CLR     CDEMRT+1
       CLR     CCOUNT
       CLR     CCOUNT+1
       JSR     TR        TRANSMIT SELF-REFERENCE
       LDA A   ECR       ENABLE CHANNEL BUFFERS
       STA A   MSKDR
ENABLE TST     MSCRB
       BPL     ENABLE
FLAG   LDA A   INTCRA    WAIT FOR INITIALIZATION FLAG TO BE
       BPL     FLAG      SET
       JSR TE
       JSR     TE        END OF TRANSMIT AND RECEIVE PHASE
       JSR     TD        SELECTION AND DECISION PROCESS
       CLR     INTFLG    CLEAR INITIALIZATION FLAG
       RTI
```

APPENDIX B

SOME CURRENT MICROPROCESSOR MANUFACTURERS

ADVANCED MICRO DEVICES
901 Thompson Rd.
Sunnyvale, CA 94086

AMERICAN MICROSYSTEMS, INC.
3800 Homestead Rd.
Santa Clara, CA 95051

FAIRCHILD SEMICONDUCTOR
464 Ellis St.
Mt View, CA 94042

GENERAL INSTRUMENTS
600 W. John St.
Hicksville, NY 11802

INTEL CORP.
3065 Bowers Ave.
Santa Clara, CA 95051

MONOLITHIC MEMORIES, INC.
1165 E. Arques Ave.
Irvine, CA 94086

MOSTEK
1215 W. Crosby Rd.
Carrollton, TX 75006

MOTOROLA SEMICONDUCTOR PRODUCTS
5005 E. McDowell
Phoenix, AZ 85062

NATIONAL SEMICONDUCTOR INC.
2900 Semiconductor
Santa Clara, CA 95051

RCA SOLID-STATE DIV.
Route 202
Somerville, NJ 08876

ROCKWELL MICROELECTRONIC DEVICE DIV.
3310 Miraloma Ave.
Anaheim, CA 92803

SIGNETICS CORP.
811 E. Arques Ave.
Sunnyvale, CA 94086

TOKYO SHIBAURA ELECTRIC CO.
Tokyo, Japan

WESTERN DIGITAL CORP.
3128 Red Hill
Newport Beach, CA 95051

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

AMERICAN REVOLUTION BICENTENNIAL
1776-1976